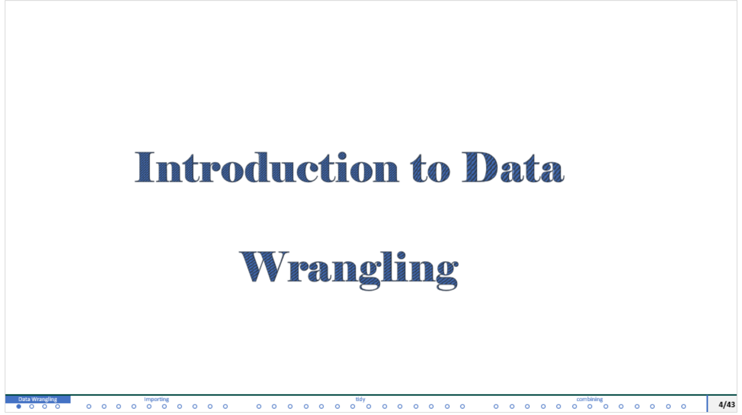
Hacettepe University
Department of Industrial Engineering
Undergraduate Program
2023-2024 Fall

EMU 430 – Data Analytics
Week 10
December 8, 2023

Instructor: Erdi Dasdemir

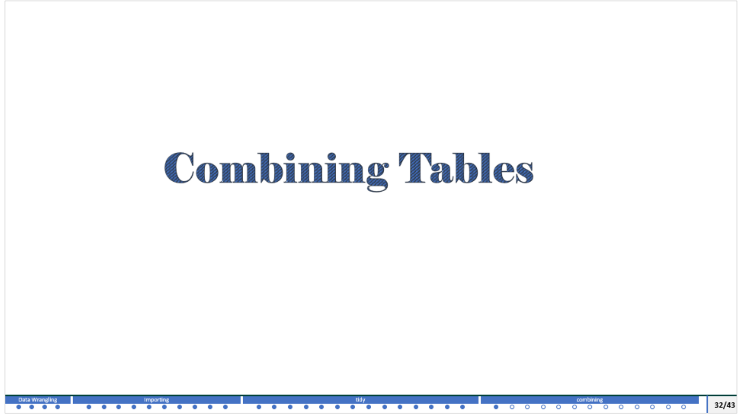
edasdemir@hacettepe.edu.tr
www.erdidasdemir.com

Outline


Introduction to Data Wrangling

Importing Spreadsheets

Tidy Your Data


Combining Tables

Acknowledgment

I drew inspiration primarily from Dr. Rafael Irizarry's "Introduction to Data Science" Book

and "Data Science" course by HarvardX on edX for the slides this week.

Introduction to Data Wrangling

Introduction to Data Wrangling

- **Data wrangling** is the process of converting raw data into a usable form.
- The data sets used in this course until now were available as data frames: the US murders data, the reported heights data, the Gapminder data…
- They are in the dslabs package, and we loaded them using the data function.
- The authors of these packages did quite a bit of work behind the scenes to get the original raw data into the tidy tables we work with.
- Yet, this is not the case in real life.

Introduction to Data Wrangling

- In a typical data science project, it is much more typical for the data to be in a file, a database, or extracted from a document, including web pages, tweets, or PDF.

- In these cases, the first step is to import the data into R, and tidy up the data.

- The first step in the data analysis process usually involves converting data from its raw form to the tidy form. We refer to this process as data wrangling.

Introduction to Data Wrangling

We will learn about common data-wrangling proocess.

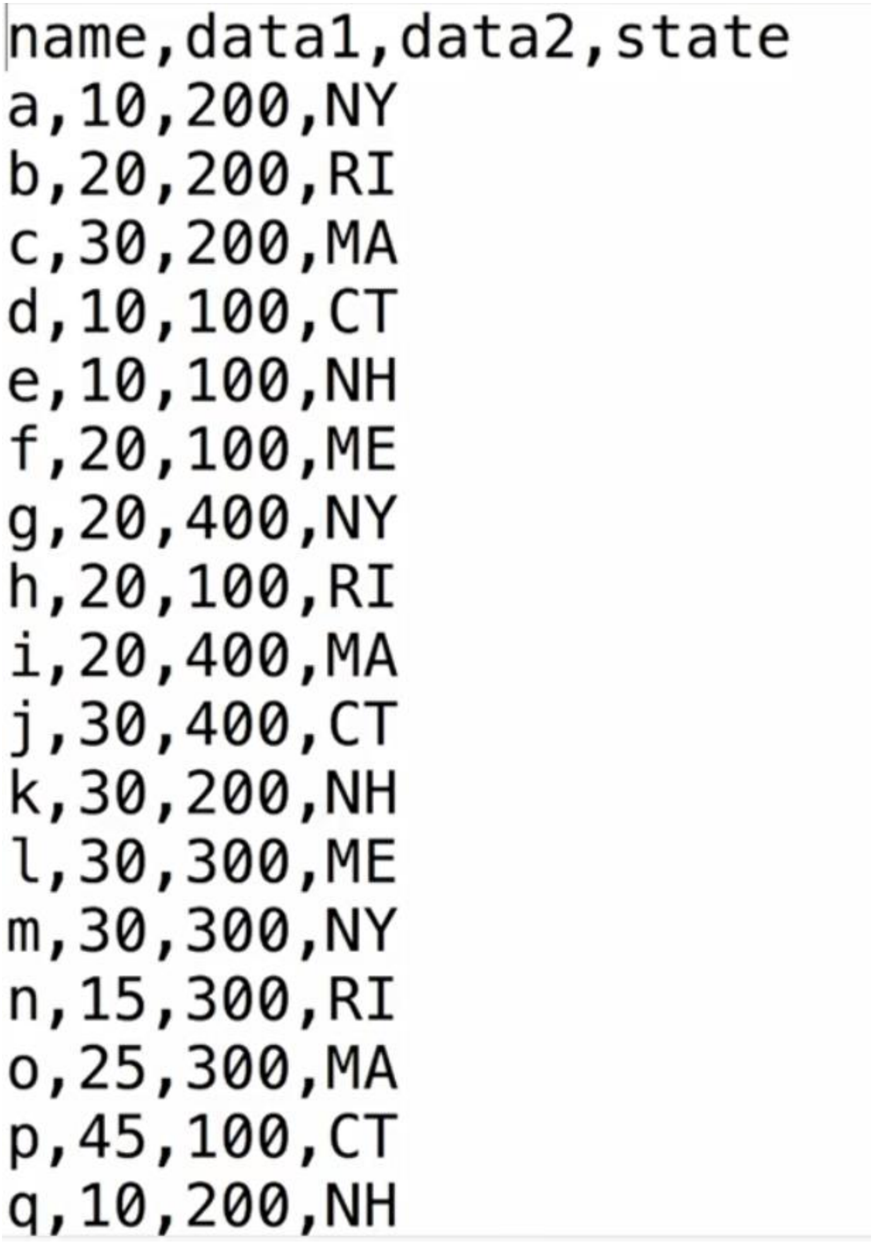
- importing data into R from files,
- tidying data,
- string processing,
- HTML parsing,
- working with dates and times, and
- text mining.

Importing Spreadsheets

Importing Spreadsheets

- A common way of storing and sharing data is through electronic spreadsheets.
- Spreadsheet: a file version of a data frame, it has rows and columns
- When creating spreadsheets that are text files,
  - new row: return
  - new column: predefined special character, the most common ones comma, semicolon, white space, tab

Example:



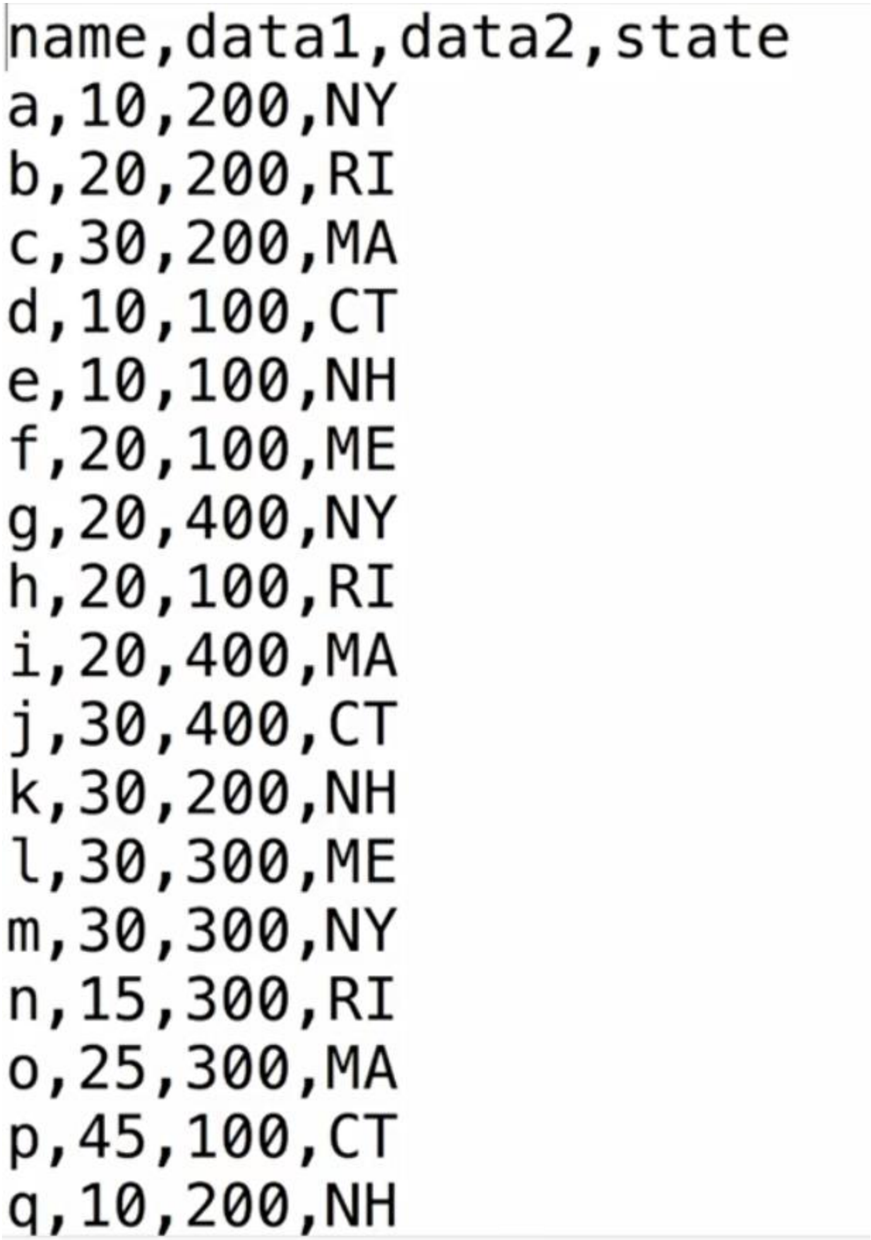
```
name,data1,data2,state
a,10,200,NY
b,20,200,RI
c,30,200,MA
d,10,100,CT
e,10,100,NH
f,20,100,ME
g,20,400,NY
h,20,100,RI
i,20,400,MA
j,30,400,CT
k,30,200,NH
l,30,300,ME
m,30,300,NY
n,15,300,RI
o,25,300,MA
p,45,100,CT
q,10,200,NH
```

Importing Spreadsheets

- Note that the first row contains column names: header

- reading from a spreadsheet, it is important to know if there is a header or not.

- Not all spreadsheet files are text files.
  - For example, Google Sheets, is rendered on a browser.
  - Microsoft Excel (we can’t see it using a text editor)



```
name,data1,data2,state
a,10,200,NY
b,20,200,RI
c,30,200,MA
d,10,100,CT
e,10,100,NH
f,20,100,ME
g,20,400,NY
h,20,100,RI
i,20,400,MA
j,30,400,CT
k,30,200,NH
l,30,300,ME
m,30,300,NY
n,15,300,RI
o,25,300,MA
p,45,100,CT
q,10,200,NH
```

Paths and Working Directory

Reading a file that is already on our computer.

- In R, it is important to know your working directory. This is the directory in which R will save or look for files by default.
- Get your working directory

```
getwd()
```

- Change your working directory

```
setwd()
```

If you are using RStudio, Session –> Set Working Directory.

- **Important:** One thing file reading functions have in common is that unless a full path is provided, they search for files in the working directory.

- **Recommendation:** Create a directory for each analysis and keep the raw data files in that directory. To make it more organized, create a data directory (folder) inside your project directory.

- **Example.** dslabs package has a raw data files as example. To find their locations:

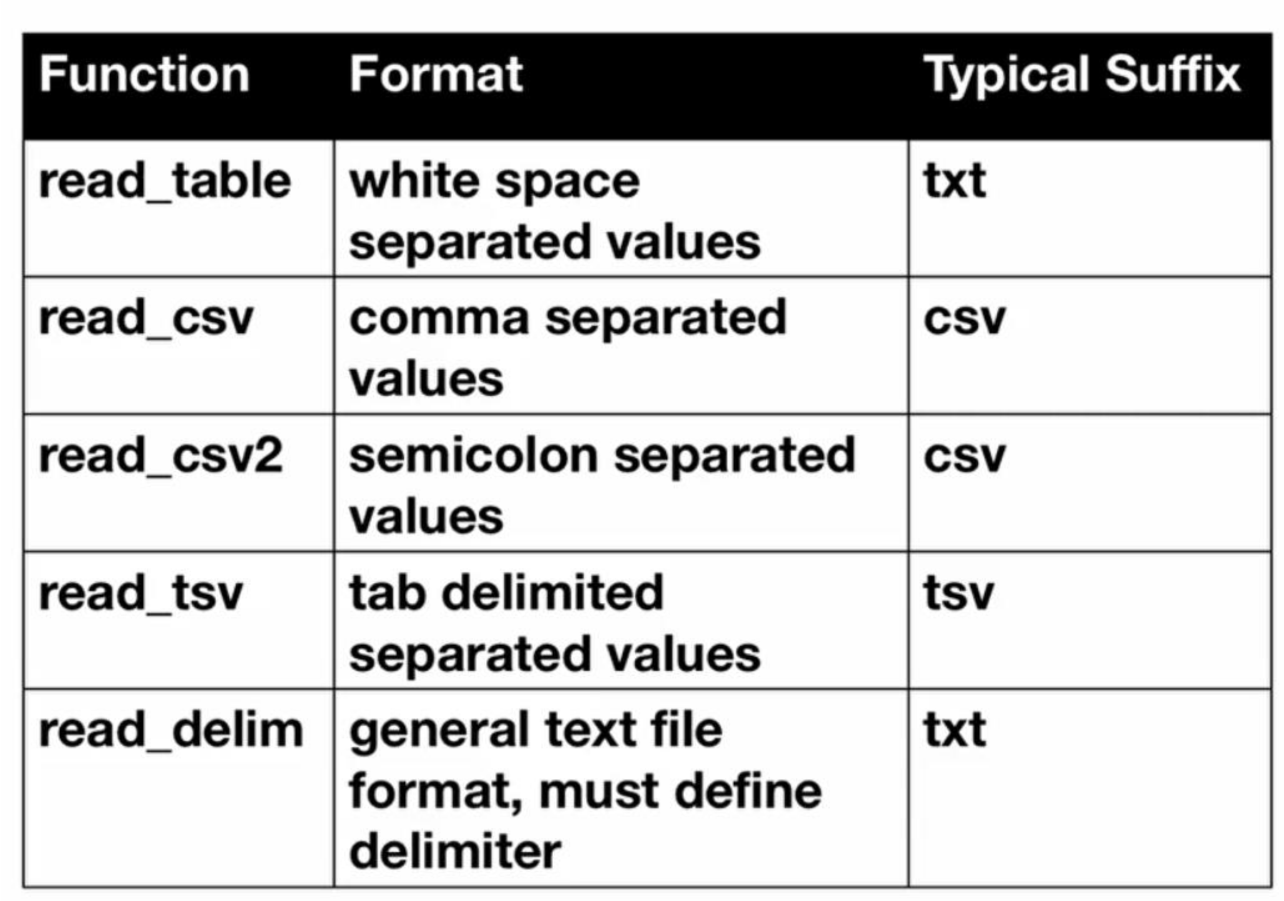
```
system.file("extdata", package = "dslabs")
```

readr and **readxl** are the tidyverse libraries that include functions for reading data stored in spreadsheets into R.

```
library(readr)
library(readxl)
```

Importing Spreadsheets — readr and readxl packages

readr

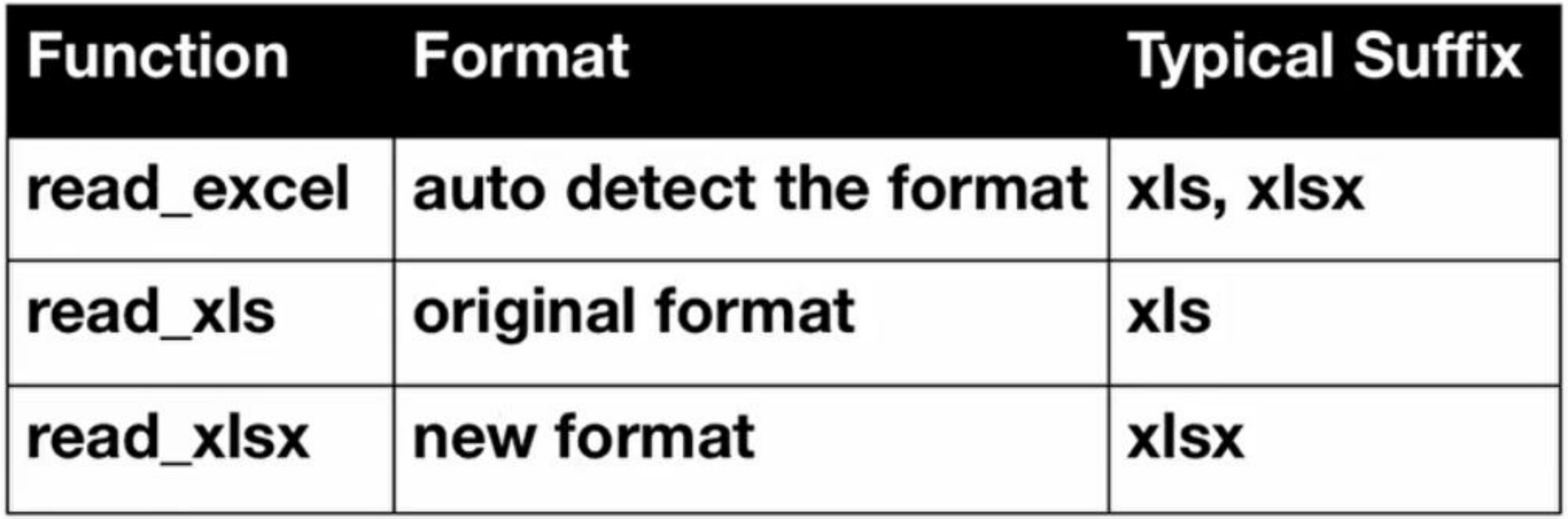


Function	Format	Typical Suffix
read_table	white space separated values	txt
read_csv	comma separated values	csv
read_csv2	semicolon separated values	csv
read_tsv	tab delimited separated values	tsv
read_delim	general text file format, must define delimiter	txt

base R functions to import data

- `read.table`
- `read.csv`
- `read.delim`

readxl



Function	Format	Typical Suffix
read_excel	auto detect the format	xls, xlsx
read_xls	original format	xls
read_xlsx	new format	xlsx

We can import or download data files from web

Example: dslabs package is on GitHub

We can download murders.csv using

```
url <-
"https://raw.githubusercontent.com/rafalab/dslabs/master/inst/extdata/murders.csv"

dat <- read_csv(url)
```

To download a local copy:

- `download.file(url, "murders.csv")`

Two functions that are sometimes useful when downloading data from the internet are

- `tempdir()`: creates a directory with a name that is unique.
- `tempfile()`: a character string, not a file, that is likely to be a unique file name;
- Download file –> give it a temporary name, read it in it, process it if needed, and then erase the file

```
tmp_filename <- tempfile()
download.file(url, tmp_filename)
dat <- read.csv(tmp_filename)
file.remove(tmp_filename)
```

Tidy Your Data

tidy your data

Example: Remember South Korea and Germany example

Once the data is proper we can use our `dplyr` and `ggplot` functions easily.

```
data("gapminder")
tidy_data <- gapminder %>% filter(country %in% c("South Korea",
"Germany")) %>% select(country, year, fertility)

head(tidy_data)
tidy_data %>% ggplot(aes(year, fertility, color = country)) +
geom_point()
```

tidy your data

- One reason the code worked seamlessly is that the data is tidy.
- Each point in the plot is represented by a row in the table.
- **tidy data:** each row represents one observation and the columns represent the different variables that we have data on for those observations.

tidy your data

Example, let’s go to the original raw version of this data file.

```
path <- system.file("extdata", package="dslabs")
filename <- file.path(path, "fertility-two-countries-example.csv")
wide_data <- read_csv(filename)
wide_data %>% select(country, '1960':'1967')
```

The data is in a wide format.

Wide Format (compared to Tidy format)

- each row includes several observations
- one of the variables is stored in the header
- ggplot does not work with wide format → we need to wrangle it to tidy format
- `tidyr` package (included in tidyverse library)

`gather()`: converts wide data into tidy data

```
help("gather")
```

- default version gathers all columns, therefore we need to specify the columns.
- we want to gather columns 1960 …. 2015

```
new_tidy_data <- wide_data %>% gather(key = year, value = 
fertility,'1960':'2015')

new_tidy_data2 <- gather(data = wide_data, key = year, value = 
fertility,'1960':'2015')

head(new_tidy_data)
```

`gather()`: converts wide data into tidy data

- we want to gather columns 1960 …. 2015
- another way to do this is
- `new_tidy_data <- wide_data %>% gather(year, fertility, -country)`
- `head(new_tidy_data)`

here is another issue: **gather function assumes column names are characters**

```
class(tidy_data$year)
[1] "integer"
class(new_tidy_data$year)
[1] "character“
```

we can use as.numeric() but gather as an argument for this.

```
new_tidy_data <- wide_data %>% gather(year, fertility, -
country, convert= TRUE)
head(new_tidy_data)
```

here is another issue: **gather function assumes column names are characters**

```
class(tidy_data$year)
[1] "integer"
class(new_tidy_data$year)
[1] "character“
```

gather has an argument for this.

```
new_tidy_data <- wide_data %>% gather(year, fertility, -country, convert=
TRUE)
class(new_tidy_data$year)
new_tidy_data  %>%  ggplot(aes(year,  fertility,  color  =  country))  +
geom_point()
```

`spread()` function

inverse of `gather()`

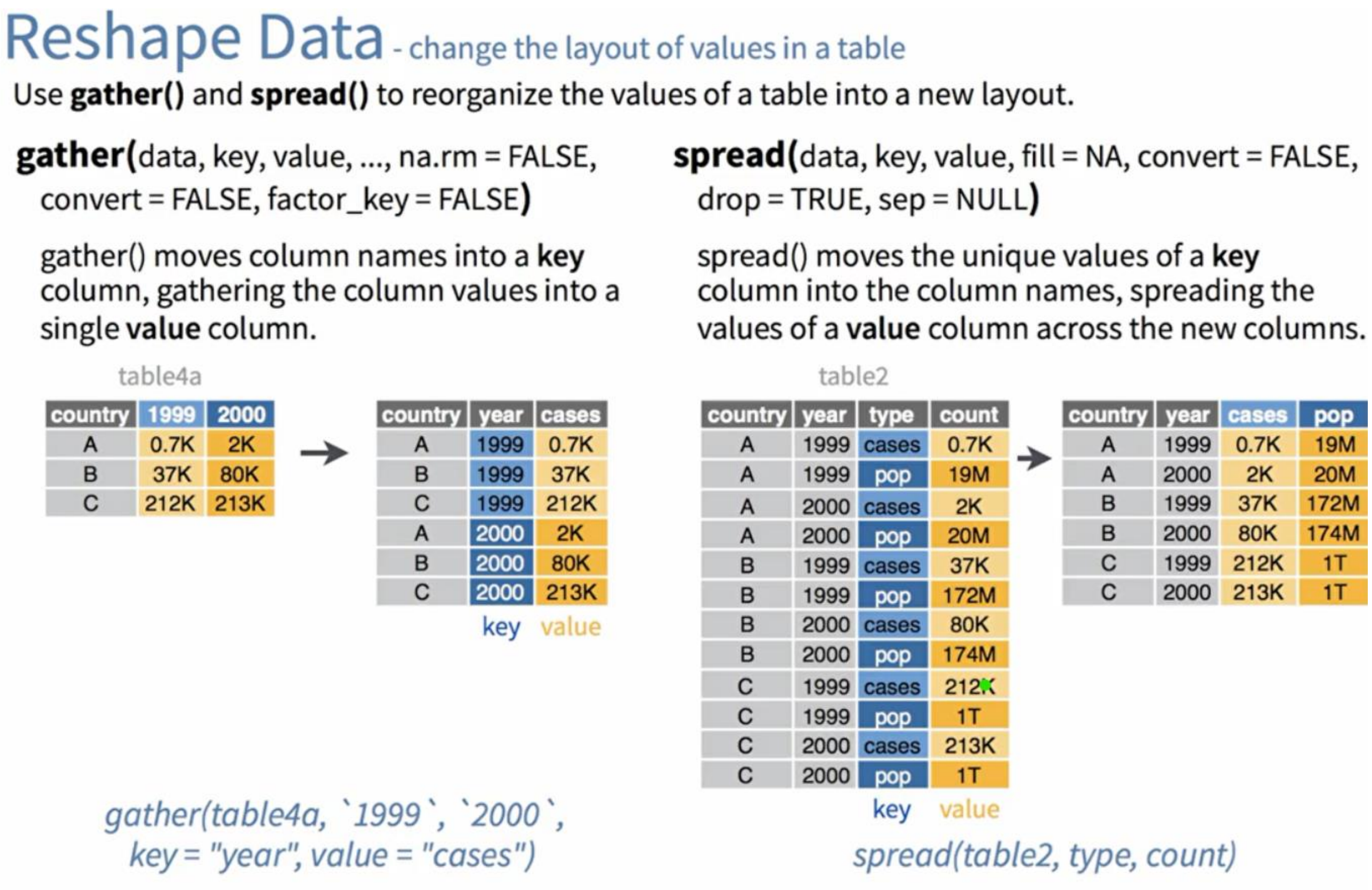
```
help(spread)
```

```
new_wide_data <- new_tidy_data %>% spread(key = year, value = fertility)
```

```
select(new_wide_data, country, '1960':'1967')
```

tidy your data tidyr package

https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf



Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout.

gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE**)**

gather() moves column names into a **key** column, gathering the column values into a single **value** column.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

key value

gather(table4a, `1999`, `2000`, key = "year", value = "cases")

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL**)**

spread() moves the unique values of a **key** column into the column names, spreading the values of a **value** column across the new columns.

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

key value

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

spread(table2, type, count)

```
path <- system.file("extdata", package="dslabs")
filename <- file.path(path, "life-expectancy-and-fertility-two-countries-
example.csv")
raw_dat <- read_csv(filename)
select(raw_dat, 1:5)
```

We will not use column name “year” as the new column name as they also include type information.

```
    dat <- raw_dat %>% gather(key, value, -country)
    head(dat)
```

Encoding multiple variables in a column name is a common problem. Hence, `readr()` has a function for this: `separate()`

```
    help(separate)
    dat %>% separate(key, c("year","variable_name"), "_")
```

tidy your data Example Case

```
path <- system.file("extdata", package="dslabs")
filename <- file.path(path, "life-expectancy-and-fertility-two-countries-example.csv")
raw_dat <- read_csv(filename)
select(raw_dat, 1:5)
```

We will not use column name “year” as the new column name as they also include type information.

```
dat <- raw_dat %>% gather(key, value, -country)
head(dat)
```

Encoding multiple variables in a column name is a common problem. Hence, `readr()` has a function for this: `separate()`

```
help(separate)
dat %>% separate(key, c("year","variable_name"), "_")
```

We are having a problem here. See the warning. “life_expectancy” is converted as “life”, as “_” is the separator.

Solution: `extra = "merge‟` argument

```
dat %>% separate(key, c("year","variable_name"), "_", extra = "merge")
```

Convert to wide format:

```
dat %>% separate(key, c("year","variable_name"), "_", extra = "merge")%>%
spread(variable_name, value) # creaates column for each variable
```

Combining Tables

Combining Tables

We may have multiple data files.

Example: We want to investigate the relationship between population and electoral votes. These are in different data sets.

```
data(murders)
head(murders)

data(polls_us_election_2016)
head(polls_us_election_2016)
```

The order of states is different in the two tables. We cannot simply put them together using column binding.

```
identical(results_us_election_2016$state, murders$state)
```

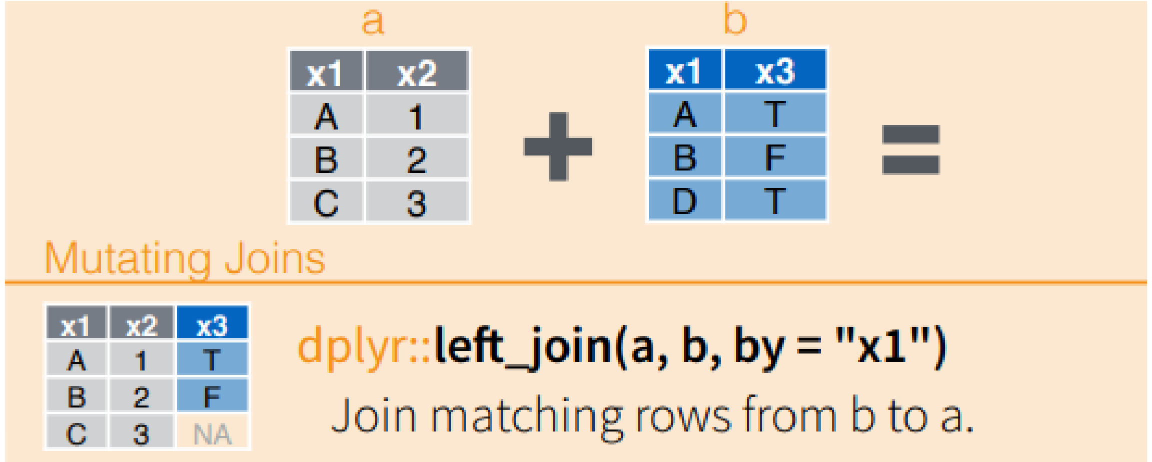
Combining Tables | join functions

These are functions from `dplyr` package:

based on SQL joins.

`left_join`

```
help("left_join")
tab <- left_join(murders,
results_us_election_2016, by =
"state")
head(tab)
```



a

x1	x2
A	1
B	2
C	3

+

b

x1	x3
A	T
B	F
D	T

=

Mutating Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NA

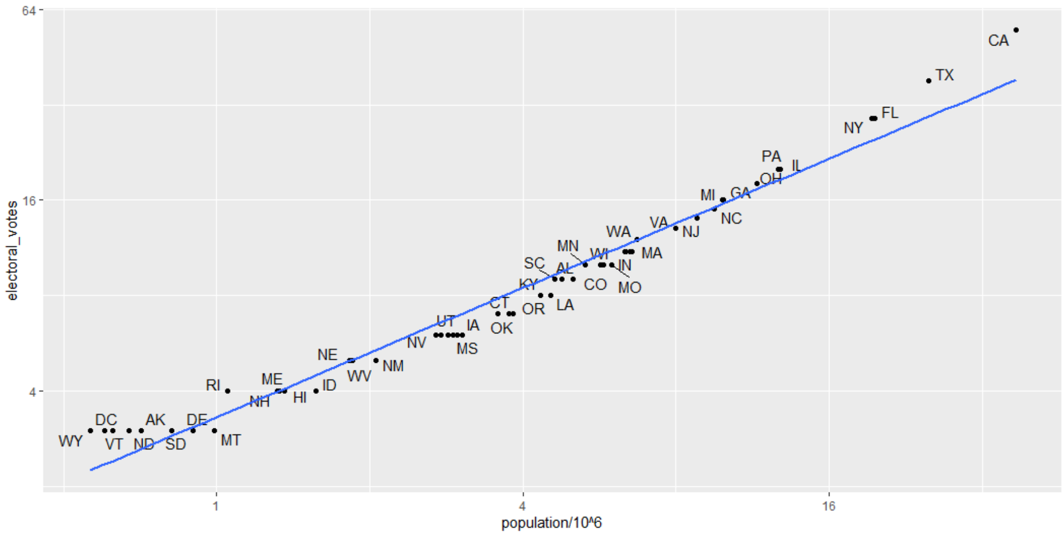
dplyr::**left_join(a, b, by = "x1")**

Join matching rows from b to a.

Combining Tables join functions

We can now make a plot to see the relationship

```
library(ggrepel)
tab %>% ggplot(aes(population/10^6,
electoral_votes, label = abb)) +
geom_point() + geom_text_repel() +
scale_x_continuous(trans = "log2") +
scale_y_continuous(trans = "log2") +
geom_smooth(method = "lm", se = FALSE)
```



- In real-life, it is not always the case that each row in one table has a matching row in the other.

Example:

```
results_us_election_2016 <- results_us_election_2016 %>%
arrange(state)

tab1 <- slice(murders, 1:6) %>%  select(state, population)
tab1

tab2 <- slice(results_us_election_2016, c(1:3, 5, 7:8)) %>%
select(state, electoral_votes)
tab2
```

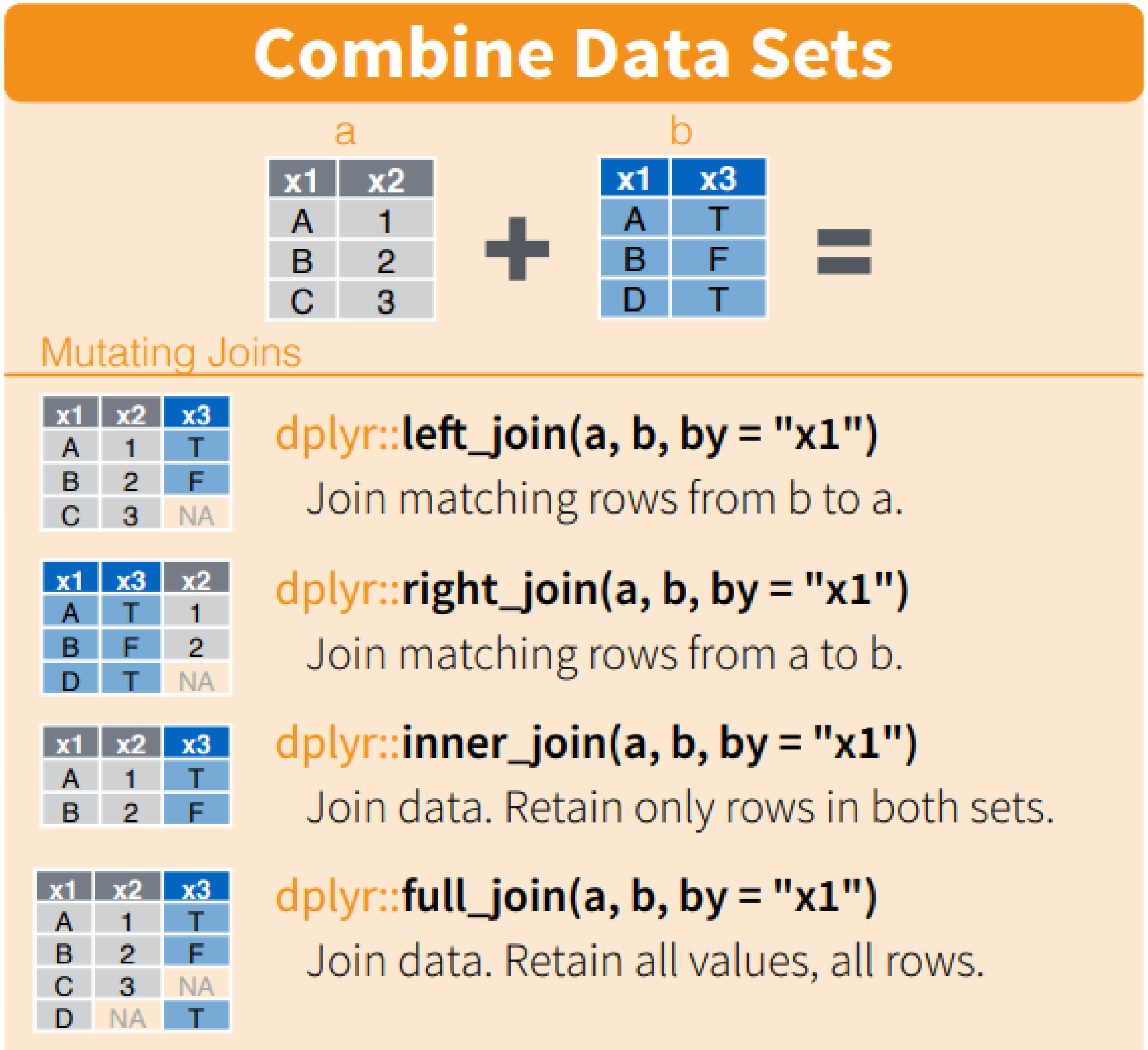
Combining Tables join functions

```
# left join
left_join(tab1, tab2)

# right join
right_join(tab1, tab2)

# keep only the rows that have
information in both tables
# inner join
inner_join(tab1, tab2) # intersection

# keep all rows and assign NAs
full_join(tab1, tab2)
```



Combine Data Sets

a

x1	x2
A	1
B	2
C	3

+

b

x1	x3
A	T
B	F
D	T

=

Mutating Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NA

dplyr::**left_join(a, b, by = "x1")**
Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

dplyr::**right_join(a, b, by = "x1")**
Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

dplyr::**inner_join(a, b, by = "x1")**
Join data. Retain only rows in both sets.

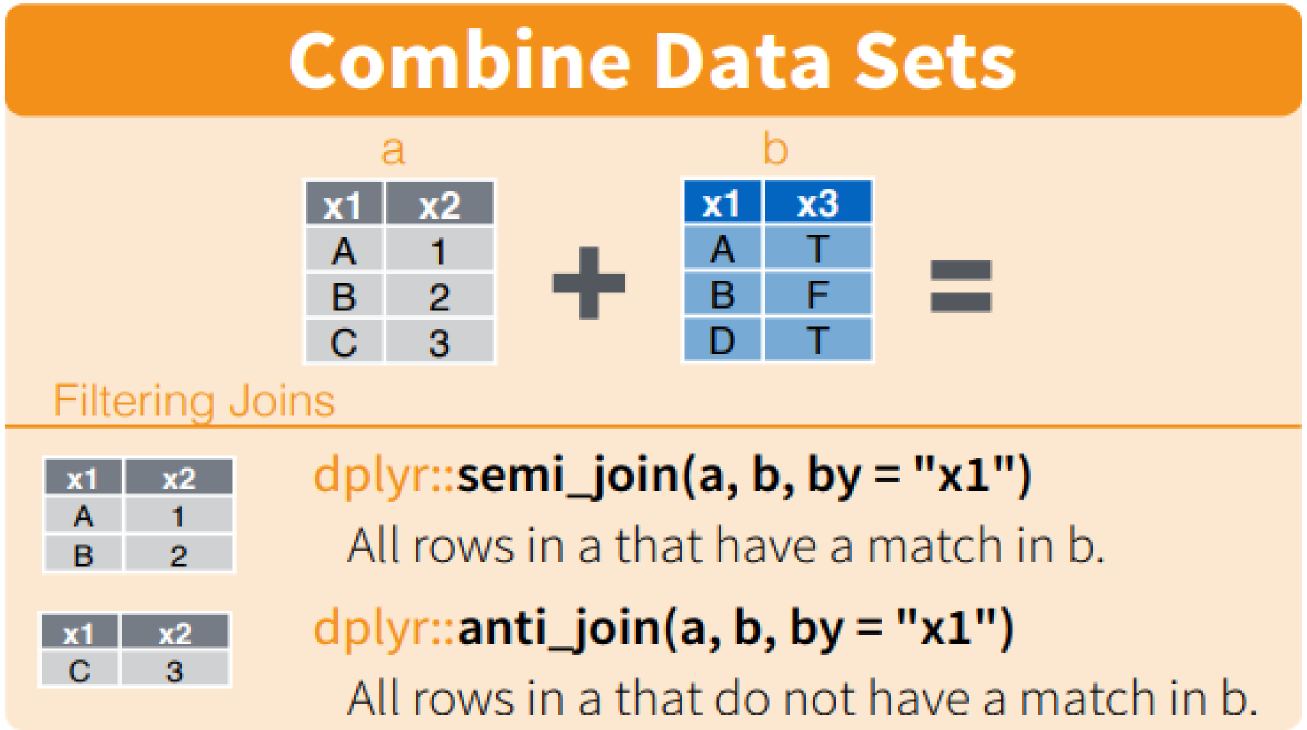
x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

dplyr::**full_join(a, b, by = "x1")**
Join data. Retain all values, all rows.

Combining Tables — join functions

```
# semi_join keeps the part of the
first table for which we have
information in the second.
semi_join(tab1, tab2)

# anti_join keeps the part of first
table for which we have no information
in the second.
anti_join(tab1, tab2)
```



Combine Data Sets

a

x1	x2
A	1
B	2
C	3

+

b

x1	x3
A	T
B	F
D	T

=

Filtering Joins

x1	x2
A	1
B	2

dplyr::semi_join(a, b, by = "x1")

All rows in a that have a match in b.

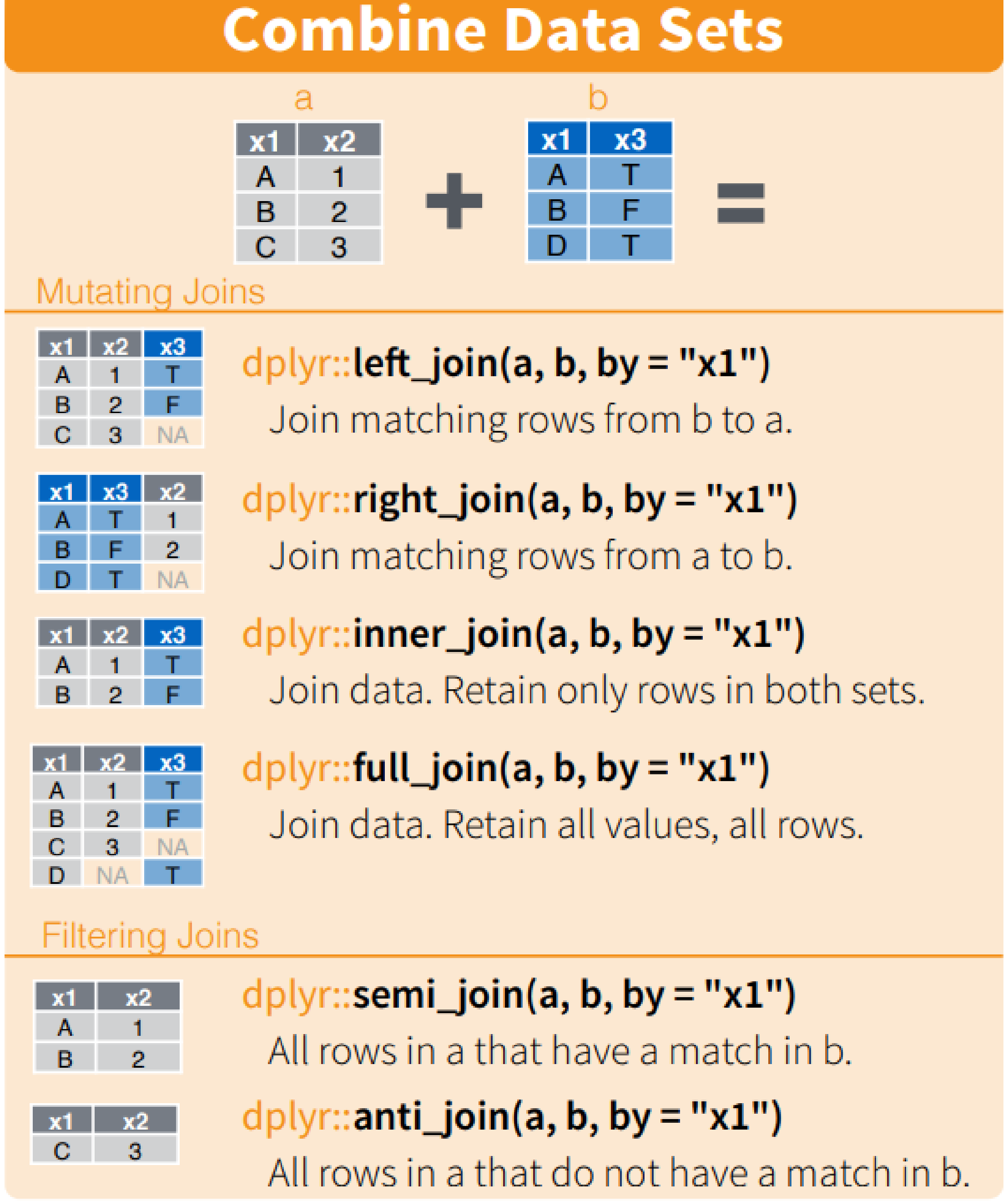
x1	x2
C	3

dplyr::anti_join(a, b, by = "x1")

All rows in a that do not have a match in b.

Combining Tables join functions

[Data Wrangling with dplyr and tidyr Cheat Sheet]



Combine Data Sets

a

x1	x2
A	1
B	2
C	3

+

b

x1	x3
A	T
B	F
D	T

=

Mutating Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NA

dplyr::**left_join(a, b, by = "x1")**

Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

dplyr::**right_join(a, b, by = "x1")**

Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

dplyr::**inner_join(a, b, by = "x1")**

Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

dplyr::**full_join(a, b, by = "x1")**

Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

dplyr::**semi_join(a, b, by = "x1")**

All rows in a that have a match in b.

x1	x2
C	3

dplyr::**anti_join(a, b, by = "x1")**

All rows in a that do not have a match in b.

dplyr has `bind_cols()`: binds two objects by putting the columns of each together in a tibble

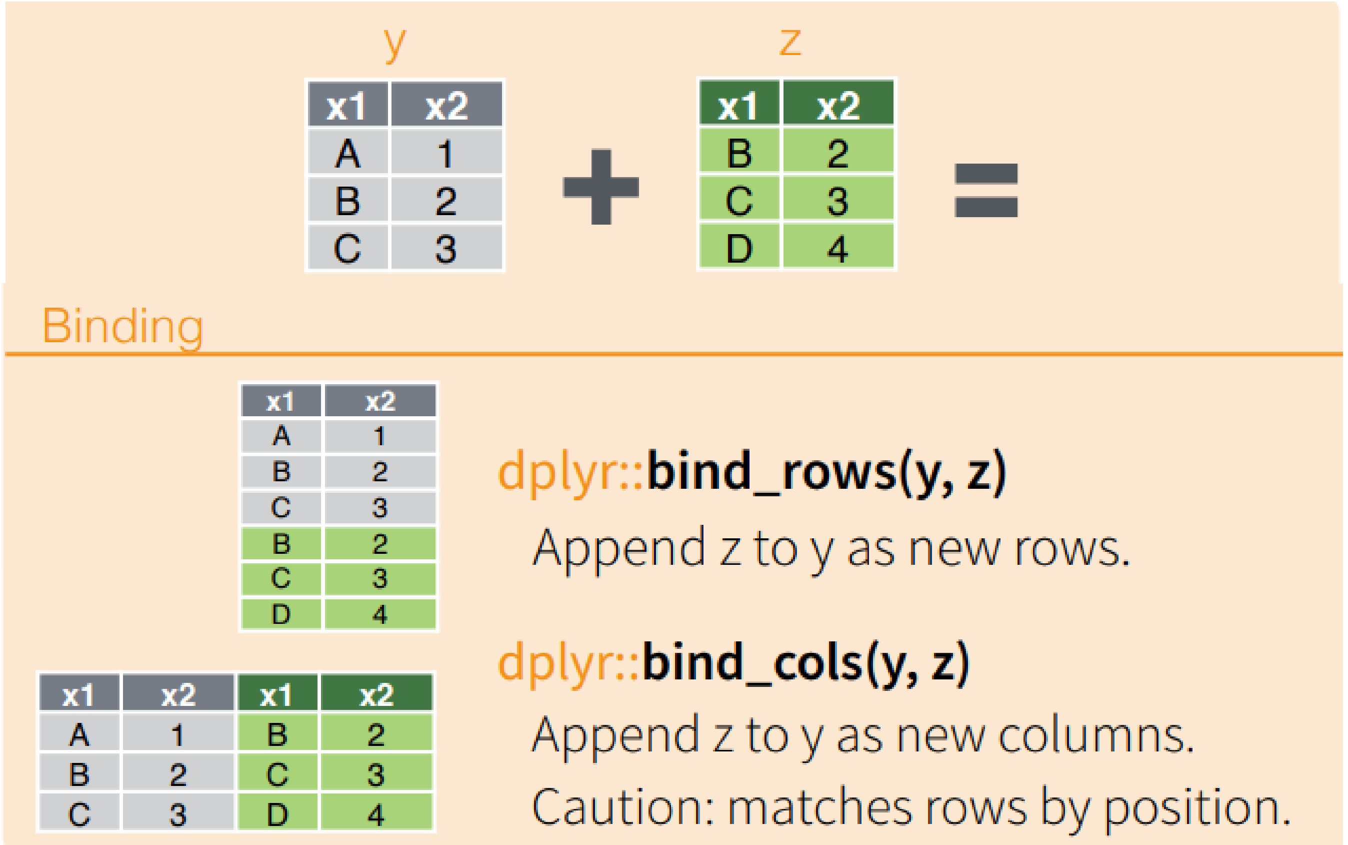
```
bind_cols(a = 1:3, b = 4:6)
```

```
cbind(a = 1:3, b = 4:6)
```

default R column binding creates objects (data frames) rather than tibbles.

We can bind data frames too

```
tab1 <- tab[, 1:3]
tab2 <- tab[, 4:6]
tab3 <- tab[, 7:9]
new_tab <-
bind_cols(tab1, tab2,
tab3)
head(new_tab)
```



y

x1	x2
A	1
B	2
C	3

+

z

x1	x2
B	2
C	3
D	4

=

Binding

x1	x2
A	1
B	2
C	3
B	2
C	3
D	4

dplyr::bind_rows(y, z)

Append z to y as new rows.

x1	x2	x1	x2
A	1	B	2
B	2	C	3
C	3	D	4

dplyr::bind_cols(y, z)

Append z to y as new columns.

Caution: matches rows by position.

Combining Tables binding functions

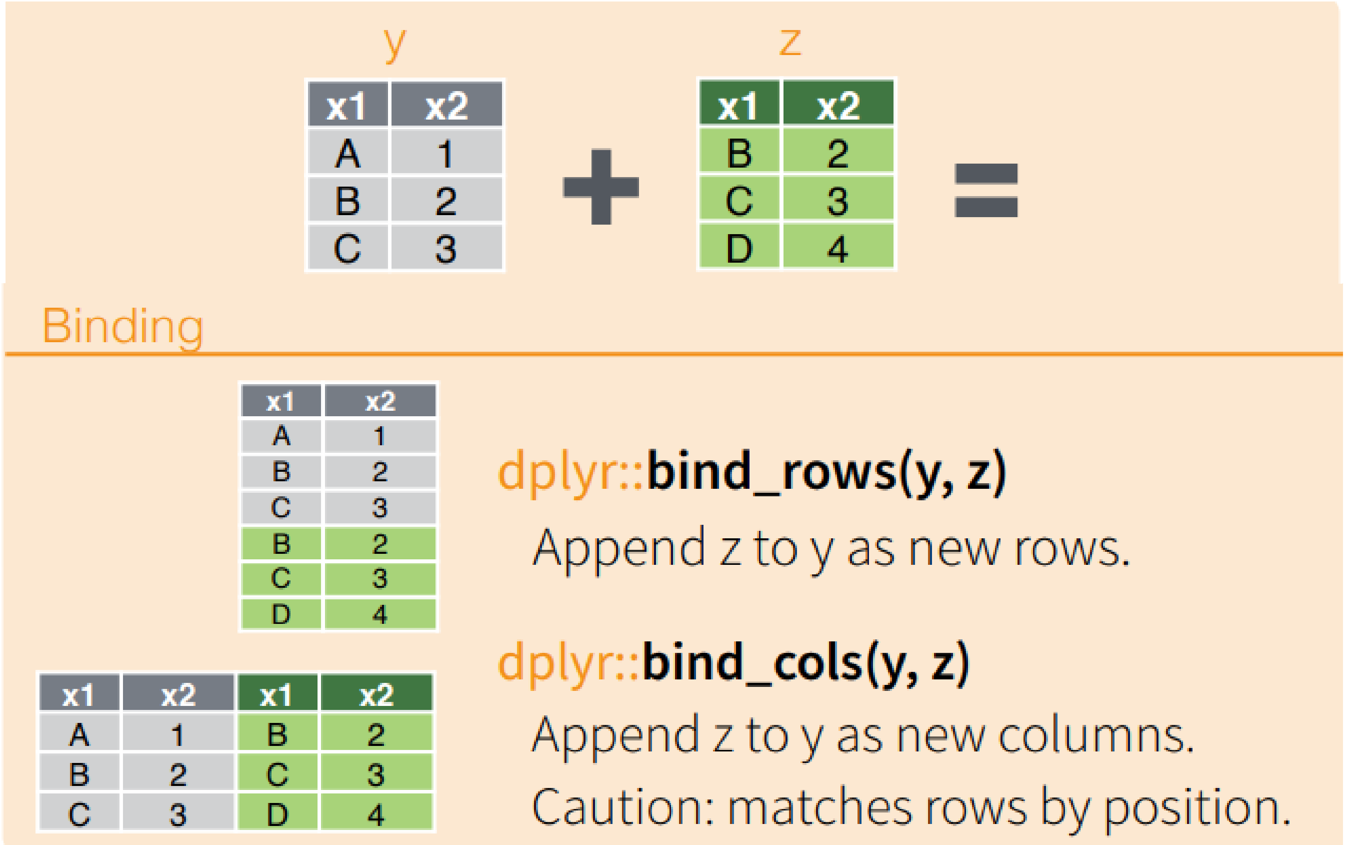
`bind_rows()` is similar but binds rows

```
tab1 <- tab[1:2, ]

tab2 <- tab[3:4, ]

bind_rows(tab1, tab2)

rbind(tab1, tab2)
```



y

x1	x2
A	1
B	2
C	3

+

z

x1	x2
B	2
C	3
D	4

=

Binding

x1	x2
A	1
B	2
C	3
B	2
C	3
D	4

dplyr::bind_rows(y, z)

Append z to y as new rows.

x1	x2	x1	x2
A	1	B	2
B	2	C	3
C	3	D	4

dplyr::bind_cols(y, z)

Append z to y as new columns.

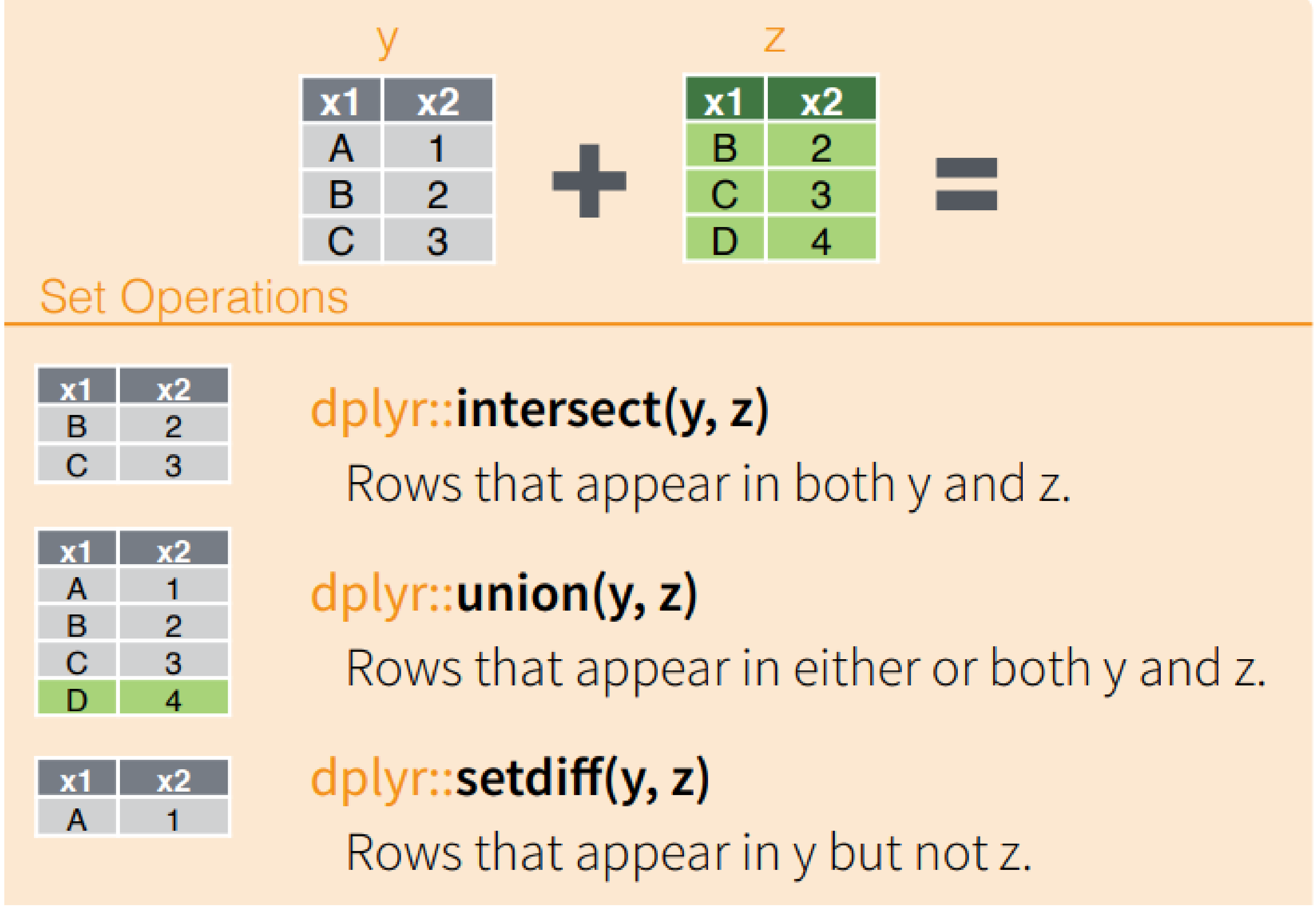
Caution: matches rows by position.

Combining Tables | set functions

```r
tab1 <- tab[1:5, ]
tab2 <- tab[3:7, ]
intersect(tab1, tab2) # intersecting rows
```

```r
tab1 <- tab[1:5, ]
tab2 <- tab[3:7, ]
union(tab1, tab2) # union rows
```

```r
tab1 <- tab[1:5, ]
tab2 <- tab[3:7, ]
setdiff(tab1, tab2) # setdiff()
```



y

x1	x2
A	1
B	2
C	3

+

z

x1	x2
B	2
C	3
D	4

=

Set Operations

x1	x2
B	2
C	3

dplyr::intersect(y, z)

Rows that appear in both y and z.

x1	x2
A	1
B	2
C	3
D	4

dplyr::union(y, z)

Rows that appear in either or both y and z.

x1	x2
A	1

dplyr::setdiff(y, z)

Rows that appear in y but not z.

Any
Questions
?