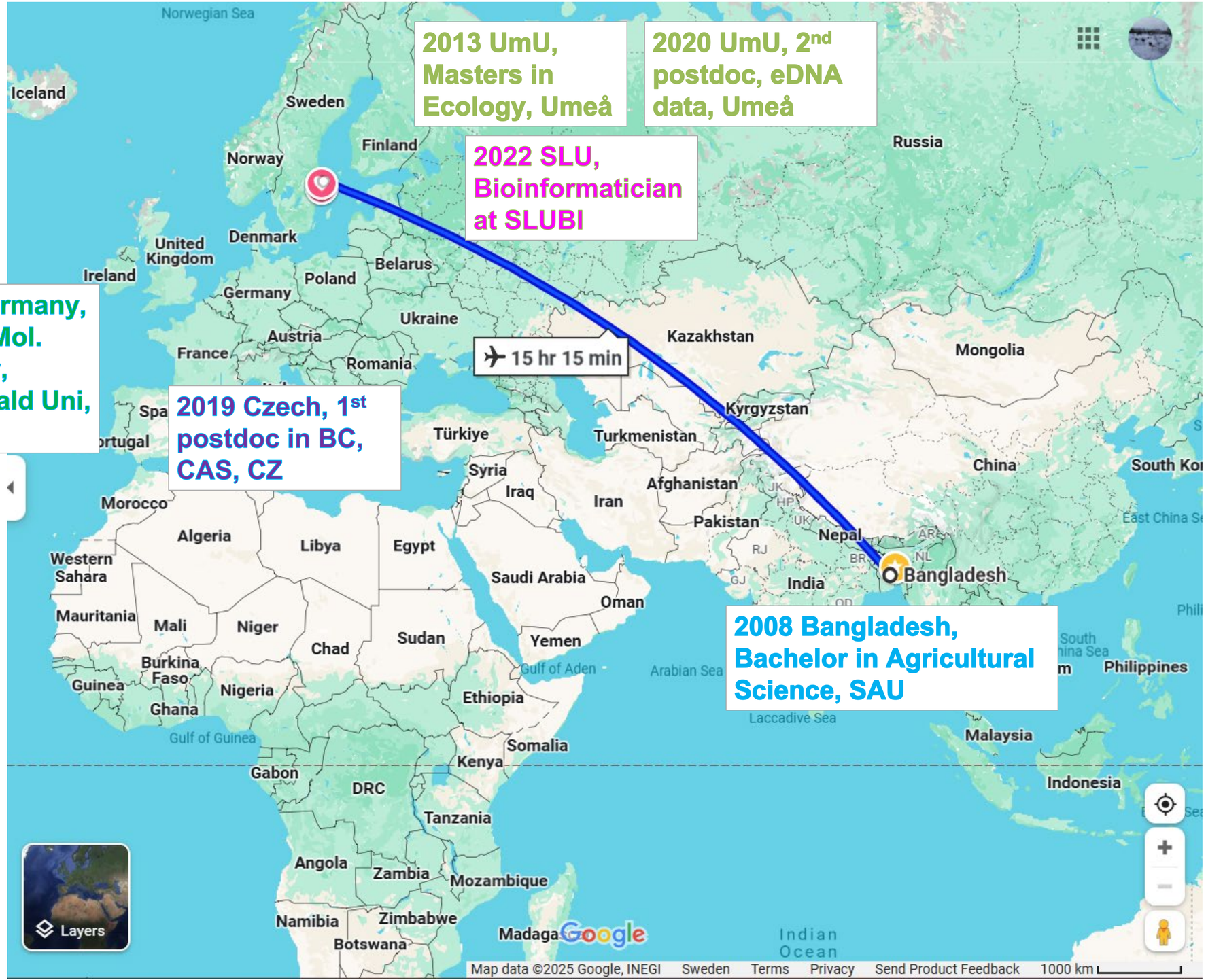
Data handling with tidyR (part 1)

Abu Bakar Siddique
SLUBI, SLU, SE

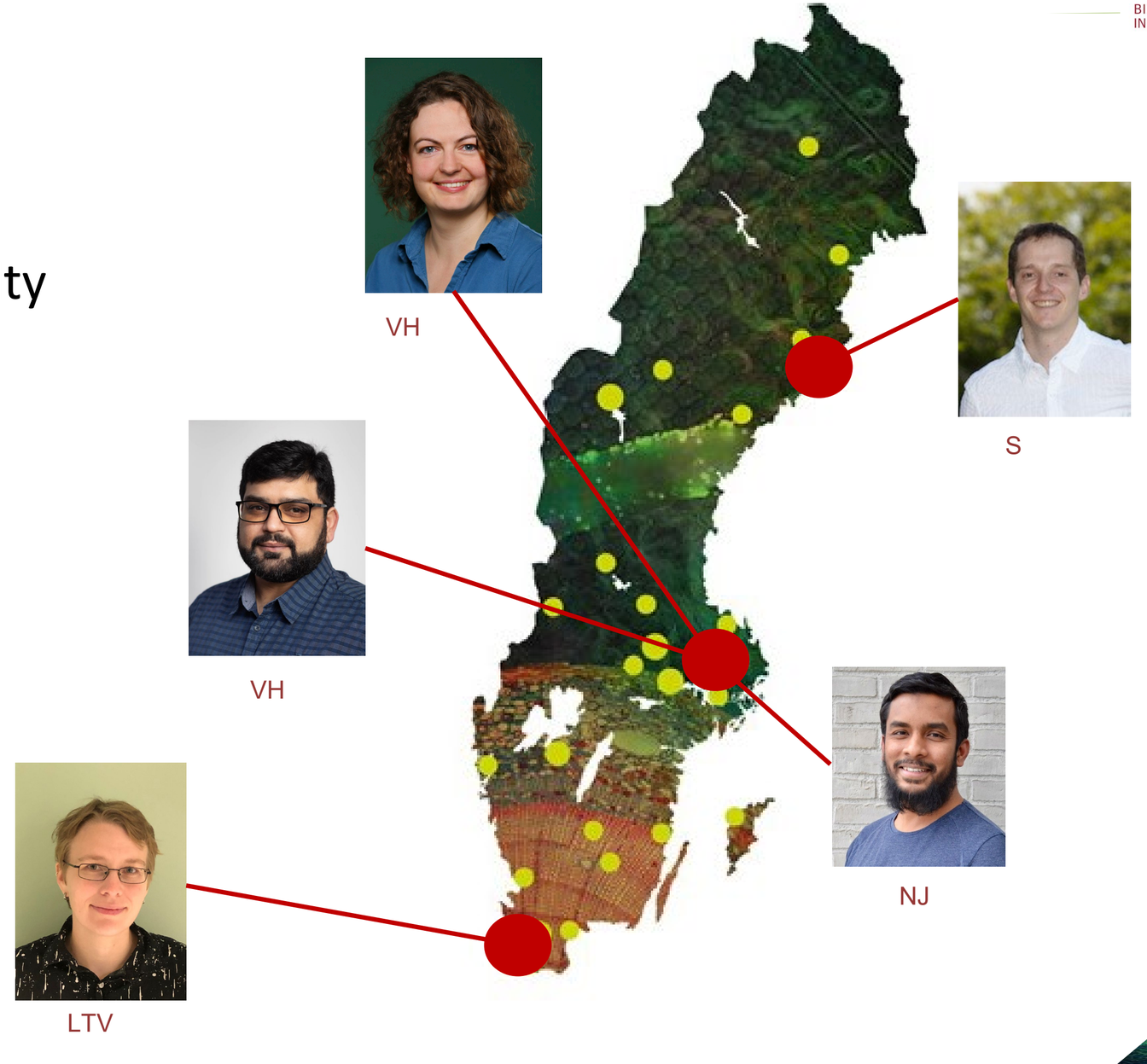
Slides adapted from: Garrett Grolemund

Me


2013 UmU, Masters in Ecology, Umeå
2020 UmU, 2nd postdoc, eDNA data, Umeå
2022 SLU, Bioinformatician at SLUBI
2017 Germany, PhD in Mol. Ecology, Greifswald Uni, DE
2019 Czech, 1st postdoc in BC, CAS, CZ
2008 Bangladesh, Bachelor in Agricultural Science, SAU
15 hr 15 min
Norwegian Sea
Iceland
Sweden
Finland
Norway
Russia
United Kingdom
Denmark
Ireland
Belarus
Poland
Germany
Ukraine
Austria
France
Romania
Kazakhstan
Mongolia
Kyrgyzstan
Türkiye
Turkmenistan
China
Syria
Iraq
Iran
Afghanistan
Pakistan
Nepal
India
Bangladesh
Morocco
Algeria
Libya
Egypt
Western Sahara
Saudi Arabia
Oman
Mauritania
Mali
Niger
Chad
Sudan
Yemen
Gulf of Aden
Arabian Sea
Burkina Faso
Guinea
Nigeria
Ghana
Ethiopia
Gulf of Guinea
Somalia
Kenya
Gabon
DRC
Tanzania
Laccadive Sea
Malaysia
Indonesia
Philippines
Angola
Zambia
Mozambique
Namibia
Zimbabwe
Botswana
Madagascar
Indian Ocean
Layers
Google
Map data ©2025 Google, INEGI
Sweden
Terms
Privacy
Send Product Feedback
1000 km

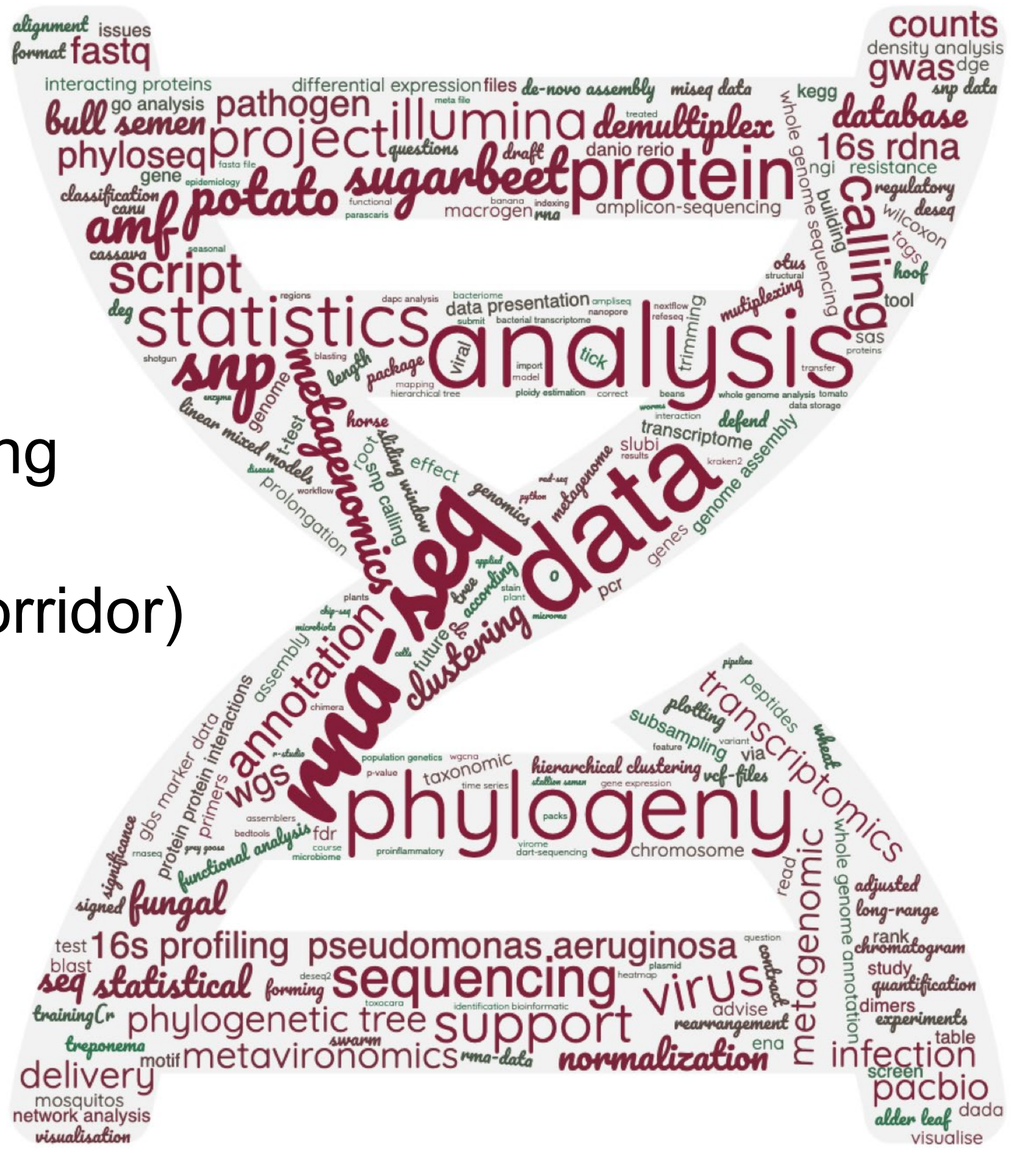
Personnel

- Expertise that matches the faculty
- Many years of experience
- Available on-site
- Network of knowledge

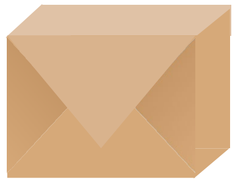


Mode of Sup

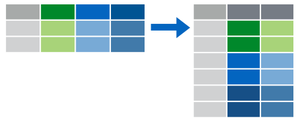
- Weekly Drop-in (ZOOM) meeting
- Individual counseling
  - ✓ Offline (in lunch break, in corridor)
  - ✓ Online (Slack, email)
- Teaching (OB: RNA, PG, GS)
- Training (Linux, UPPMAX, R)
- **Contract work (Project)**



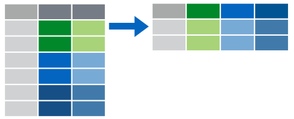
tidyr



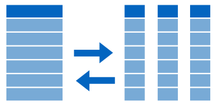
A package that reshapes the layout of data sets.



Make observations from variables with `pivot_longer()`



Make variables from observations with `pivot_wider()`



Split and merge columns with `separate()` and `unite()`

Why?

Figure: Alison Horst

Why?

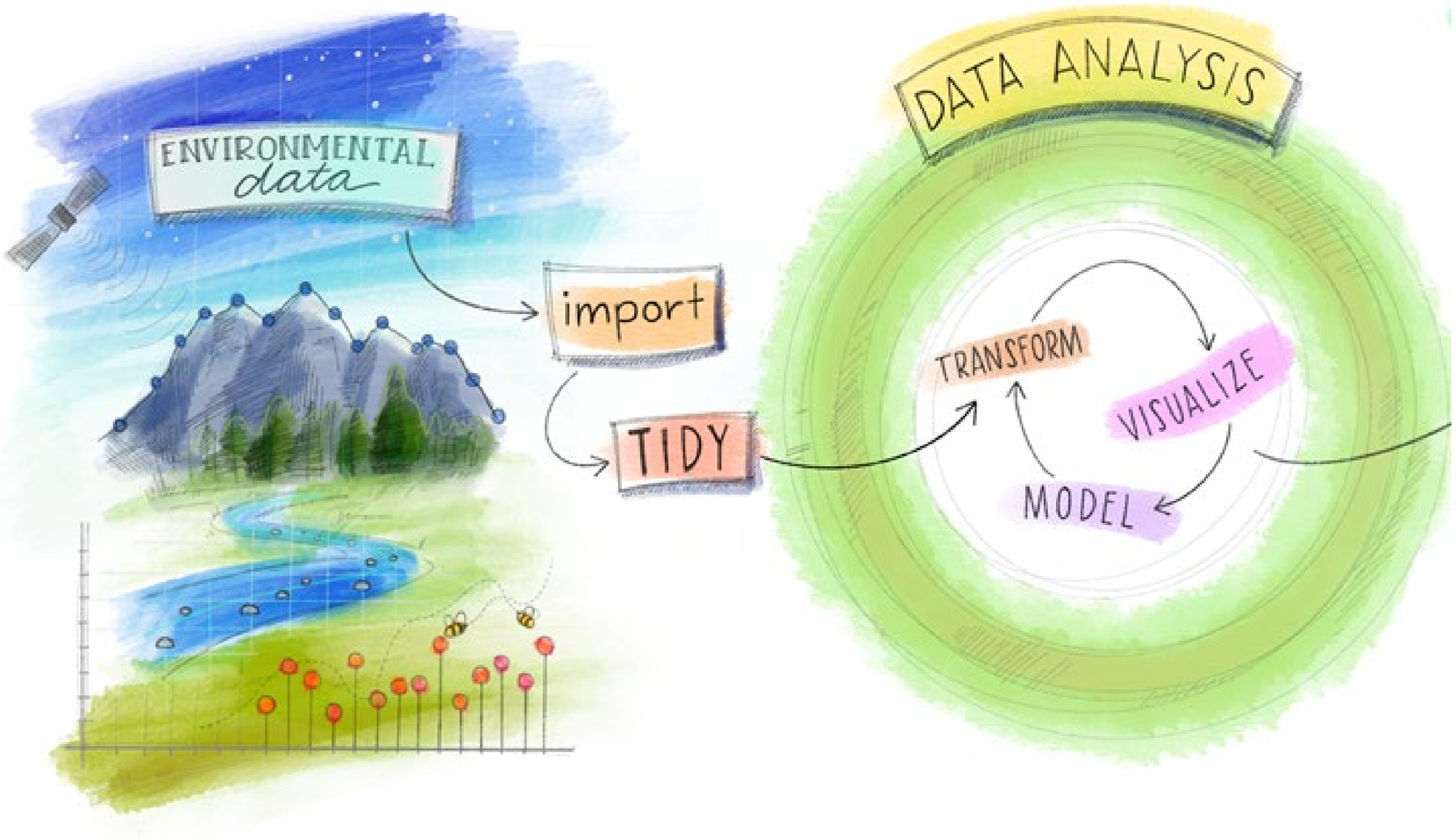


Figure: Alison Horst

Why?

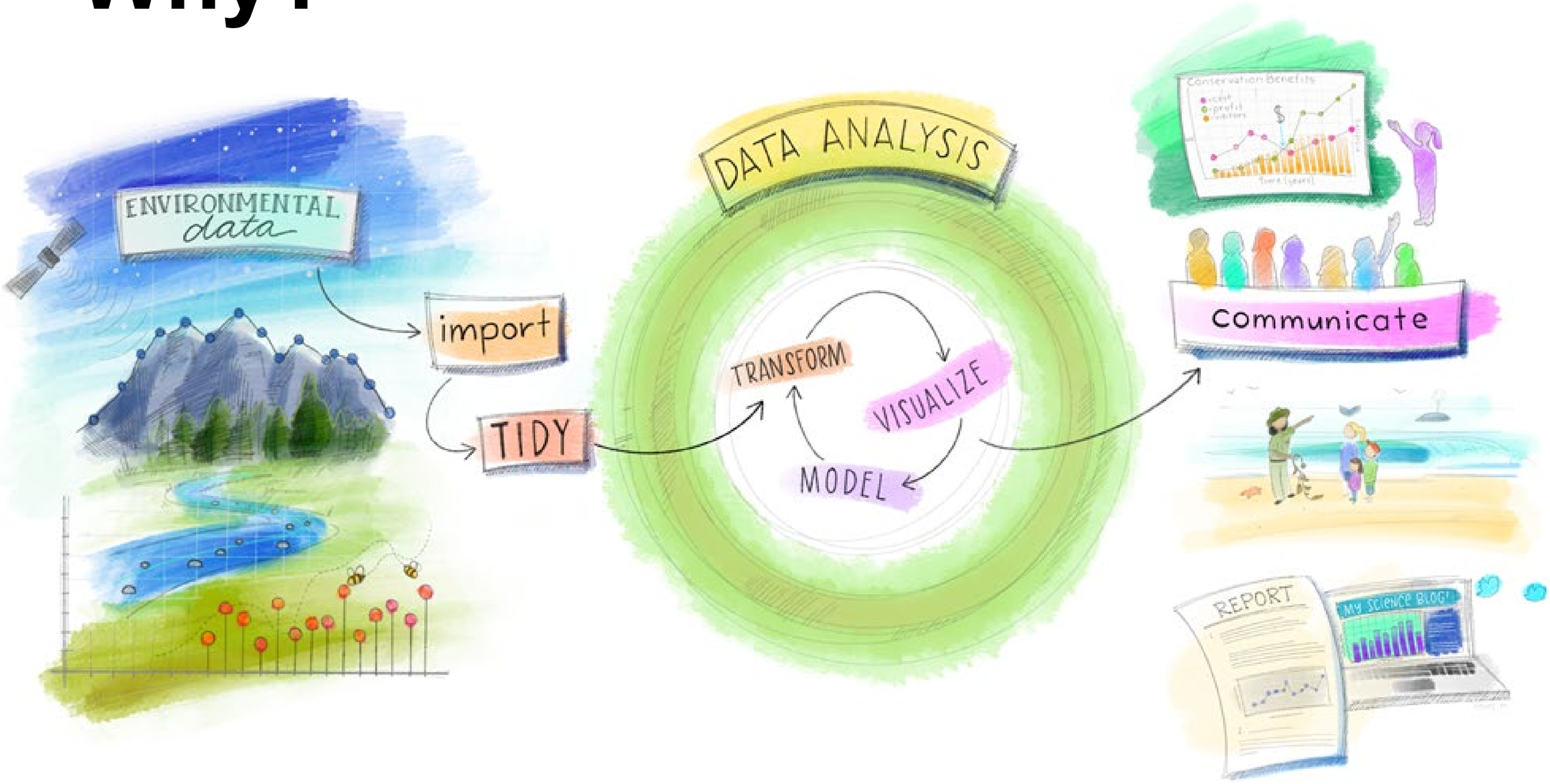
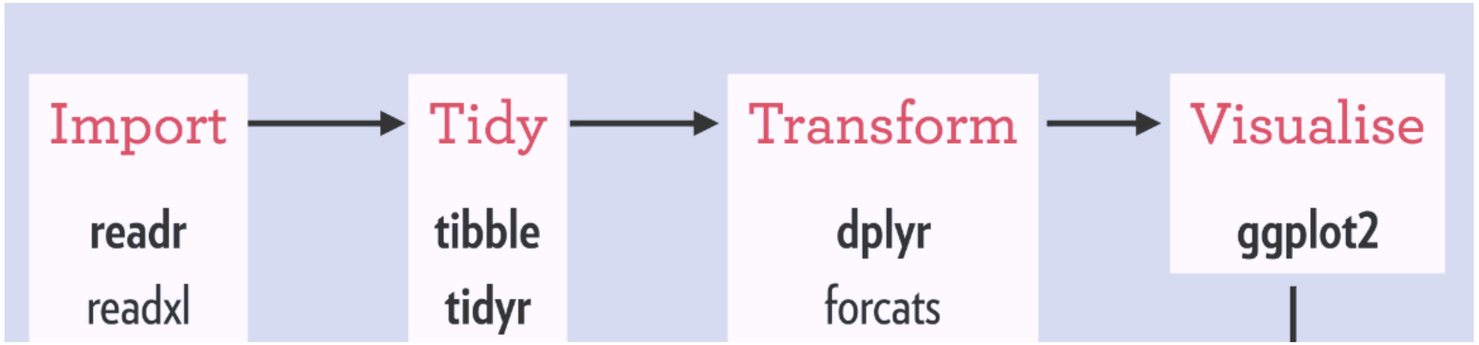


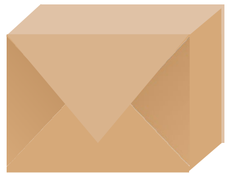
Figure: Alison Horst

How?

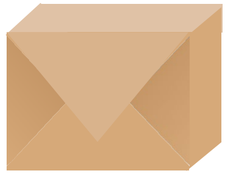
Tidyverse package in R



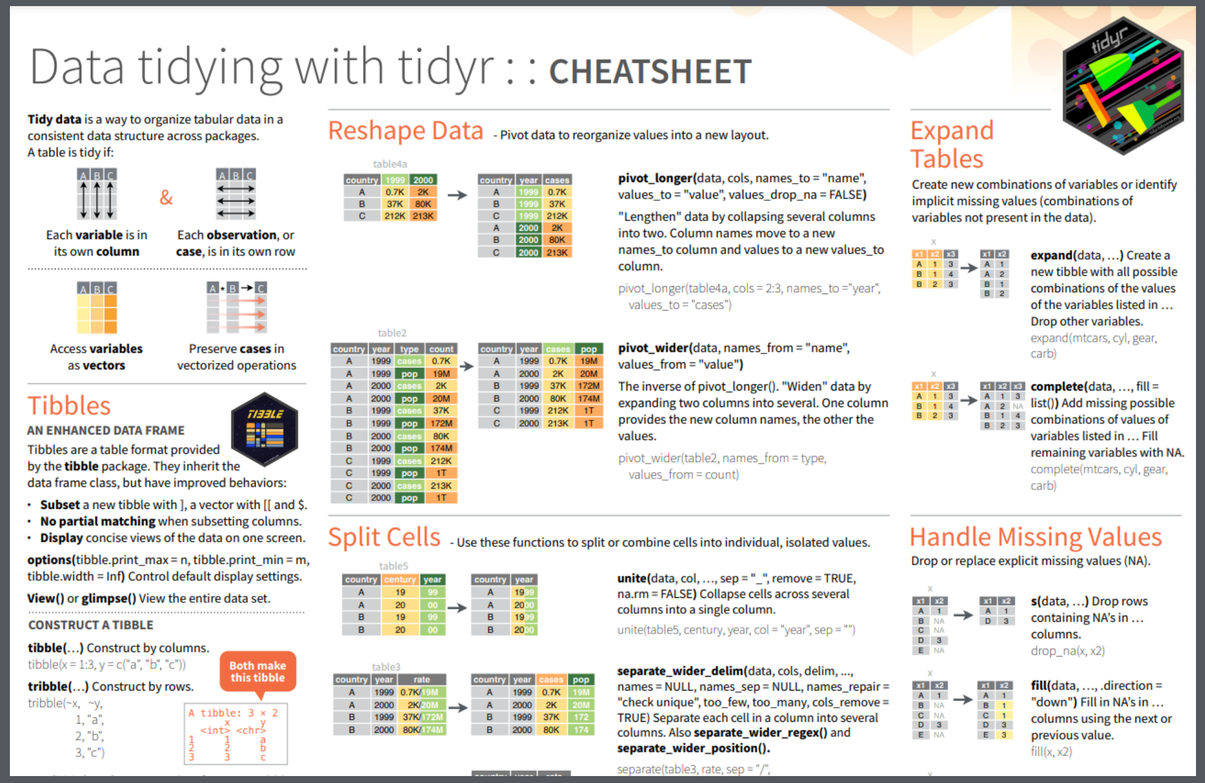
Two packages to help us
work with the structure of data.



tidyr



dplyr



Data tidying with tidyr : : CHEATSHEET

Tidy data is a way to organize tabular data in a consistent data structure across packages.
A table is tidy if:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own row

Access **variables** as **vectors** — Preserve **cases** in vectorized operations

Tibbles

AN ENHANCED DATA FRAME

Tibbles are a table format provided by the **tibble** package. They inherit the data frame class, but have improved behaviors:

- **Subset** a new tibble with], a vector with [[and $.
- **No partial matching** when subsetting columns.
- **Display** concise views of the data on one screen.

options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf) Control default display settings.

View() or **glimpse()** View the entire data set.

CONSTRUCT A TIBBLE

tibble(…) Construct by columns.
tibble(x = 1:3, y = c("a", "b", "c"))

tribble(…) Construct by rows.
tribble(~x, ~y,
1, "a",
2, "b",
3, "c")

Both make this tibble

A tibble: 3 × 2

x	y
<int>	<chr>
1	a
2	b
3	c

Reshape Data - Pivot data to reorganize values into a new layout.

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

pivot_longer(data, cols, names_to = "name", values_to = "value", values_drop_na = FALSE)

"Lengthen" data by collapsing several columns into two. Column names move to a new names_to column and values to a new values_to column.

pivot_longer(table4a, cols = 2:3, names_to ="year", values_to = "cases")

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

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

pivot_wider(data, names_from = "name", values_from = "value")

The inverse of pivot_longer(). "Widen" data by expanding two columns into several. One column provides the new column names, the other the values.

pivot_wider(table2, names_from = type, values_from = count)

Split Cells - Use these functions to split or combine cells into individual, isolated values.

table5

country	century	year
A	19	99
A	20	00
B	19	99
B	20	00

country	year
A	1999
A	2000
B	1999
B	2000

unite(data, col, …, sep = "_", remove = TRUE, na.rm = FALSE) Collapse cells across several columns into a single column.

unite(table5, century, year, col = "year", sep = "")

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174

separate_wider_delim(data, cols, delim, ..., names = NULL, names_sep = NULL, names_repair = "check unique", too_few, too_many, cols_remove = TRUE) Separate each cell in a column into several columns. Also **separate_wider_regex()** and **separate_wider_position()**.

separate(table3, rate, sep = "/",

Expand Tables

Create new combinations of variables or identify implicit missing values (combinations of variables not present in the data).

expand(data, …) Create a new tibble with all possible combinations of the values of the variables listed in … Drop other variables.
expand(mtcars, cyl, gear, carb)

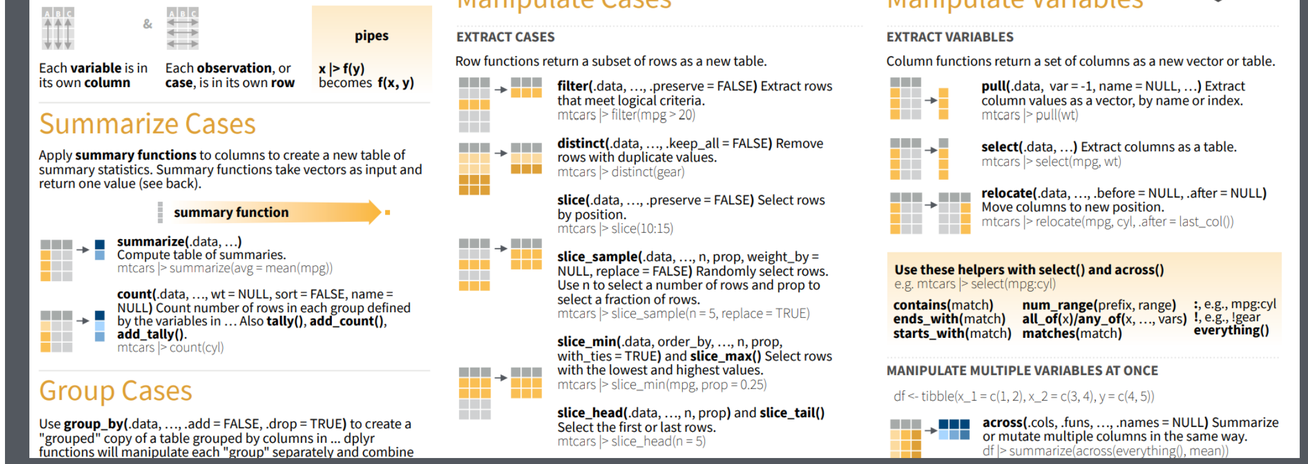
complete(data, …, fill = list()) Add missing possible combinations of values of variables listed in … Fill remaining variables with NA.
complete(mtcars, cyl, gear, carb)

Handle Missing Values

Drop or replace explicit missing values (NA).

s(data, …) Drop rows containing NA's in … columns.
drop_na(x, x2)

fill(data, …, .direction = "down") Fill in NA's in … columns using the next or previous value.
fill(x, x2)



Data transformation with dplyr : : CHEATSHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

pipes
x |> f(y) becomes **f(x, y)**

Summarize Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarize(.data, …)
Compute table of summaries.
mtcars |> summarize(avg = mean(mpg))

count(.data, …, wt = NULL, sort = FALSE, name = NULL) Count number of rows in each group defined by the variables in … Also **tally()**, **add_count()**, **add_tally()**.
mtcars |> count(cyl)

Group Cases

Use **group_by**(.data, …, .add = FALSE, .drop = TRUE) to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …, .preserve = FALSE) Extract rows that meet logical criteria.
mtcars |> filter(mpg > 20)

distinct(.data, …, .keep_all = FALSE) Remove rows with duplicate values.
mtcars |> distinct(gear)

slice(.data, …, .preserve = FALSE) Select rows by position.
mtcars |> slice(10:15)

slice_sample(.data, …, n, prop, weight_by = NULL, replace = FALSE) Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.
mtcars |> slice_sample(n = 5, replace = TRUE)

slice_min(.data, order_by, …, n, prop, with_ties = TRUE) and **slice_max()** Select rows with the lowest and highest values.
mtcars |> slice_min(mpg, prop = 0.25)

slice_head(.data, …, n, prop) and **slice_tail()** Select the first or last rows.
mtcars |> slice_head(n = 5)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1, name = NULL, …) Extract column values as a vector, by name or index.
mtcars |> pull(wt)

select(.data, …) Extract columns as a table.
mtcars |> select(mpg, wt)

relocate(.data, …, .before = NULL, .after = NULL) Move columns to new position.
mtcars |> relocate(mpg, cyl, .after = last_col())

Use these helpers with select() and across()
e.g. mtcars |> select(mpg:cyl)

contains(match)
ends_with(match)
starts_with(match)
num_range(prefix, range)
all_of(x)/**any_of**(x, …, vars)
matches(match)
:, e.g., mpg:cyl
!, e.g., !gear
everything()

MANIPULATE MULTIPLE VARIABLES AT ONCE

df <- tibble(x_1 = c(1, 2), x_2 = c(3, 4), y = c(4, 5))

across(.cols, .funs, …, .names = NULL) Summarize or mutate multiple columns in the same way.
df |> summarize(across(everything(), mean))

https://rstudio.github.io/cheatsheets/

Ground rules

tbl's

Just like data frames, but play better with the console window.

```
Source: local data frame [53,940 x 10]

   carat       cut color clarity depth table
1   0.23     Ideal     E     SI2  61.5    55
2   0.21   Premium     E     SI1  59.8    61
3   0.23      Good     E     VS1  56.9    65
4   0.29   Premium     I     VS2  62.4    58
5   0.31      Good     J     SI2  63.3    58
6   0.24 Very Good     J    VVS2  62.8    57
7   0.24 Very Good     I    VVS1  62.3    57
8   0.26 Very Good     H     SI1  61.9    55
9   0.22      Fair     E     VS2  65.1    61
10  0.23 Very Good     H     VS1  59.4    61
..   ...       ...   ...     ...   ...   ...
Variables not shown: price (int), x (dbl), y
  (dbl), z (dbl)
```

tbl

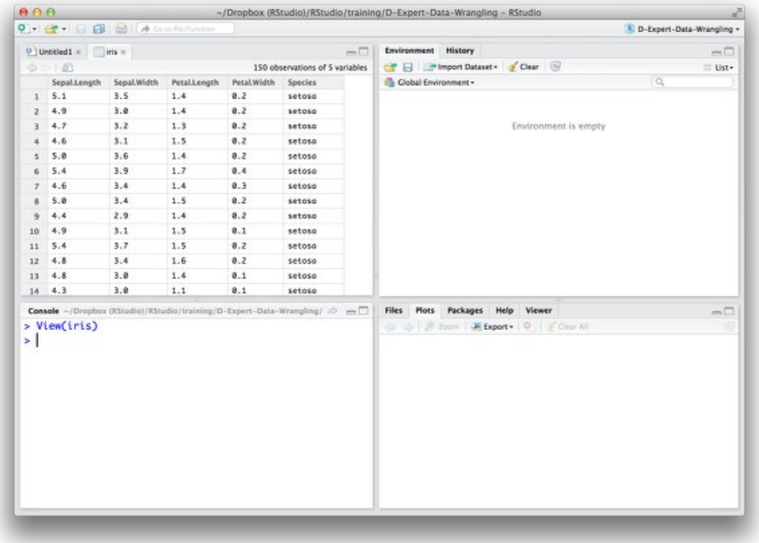
```
977     59.0  2893  6.09  6.06  3.64
978     57.0  2894  5.91  5.99  3.71
979     57.0  2894  5.96  6.00  3.72
980     56.0  2894  5.88  5.92  3.62
981     56.0  2895  5.75  5.78  3.51
982     59.0  2895  5.66  5.76  3.53
983     53.0  2895  5.71  5.75  3.56
986     63.0  2896  6.00  6.05  3.51
987     56.0  2896  5.18  5.24  3.21
988     56.0  2896  5.91  5.96  3.65
989     55.0  2896  5.82  5.86  3.59
990     56.0  2896  5.83  5.89  3.64
991     58.0  2896  5.94  5.88  3.60
992     57.0  2896  6.39  6.35  4.02
993     57.0  2896  6.46  6.45  3.97
994     57.0  2897  5.48  5.51  3.33
995     58.0  2897  5.91  5.85  3.59
996     52.0  2897  5.30  5.34  3.26
997     55.0  2897  5.69  5.74  3.57
998     61.0  2897  5.82  5.89  3.48
999     58.0  2897  5.81  5.77  3.58
1000    59.0  2898  6.68  6.61  4.03
 [ reached getOption("max.print") --
omitted 52940 rows ]
```

data.frame

View()

Examine any data set with the View() command (Capital V)

```
View(iris)
View(mtcars)
View(pressure)
```



Data Wrangling

Wrangling
Munging
Janitor Work
Manipulation
Transformation

50-80%
of our time?

Two goals

1 **Make data suitable** to use with a particular piece of software

2 Reveal information

Data sets come in many formats

…but R prefers just one.

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

cases

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

pollution

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

https://github.com/rstudio/EDAWR/tree/master/data-raw

Uploading a data

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

```
install.packages("readr")
library(readr)# package

cases <- read_csv(
"https://github.com/rstudio/EDAWR/blob/mast
er/data-raw/cases.csv?raw=true")

View(cases)
```

replace blue texts with '*pollution*' or '*storms*' or '*tb*' data

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

cases

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

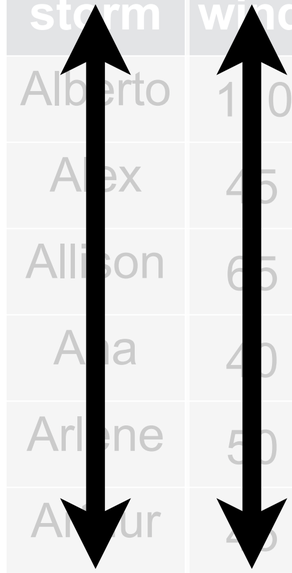
pollution

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

- Storm name

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



cases

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

pollution

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

- Storm name
- Wind Speed (mph)

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

cases

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

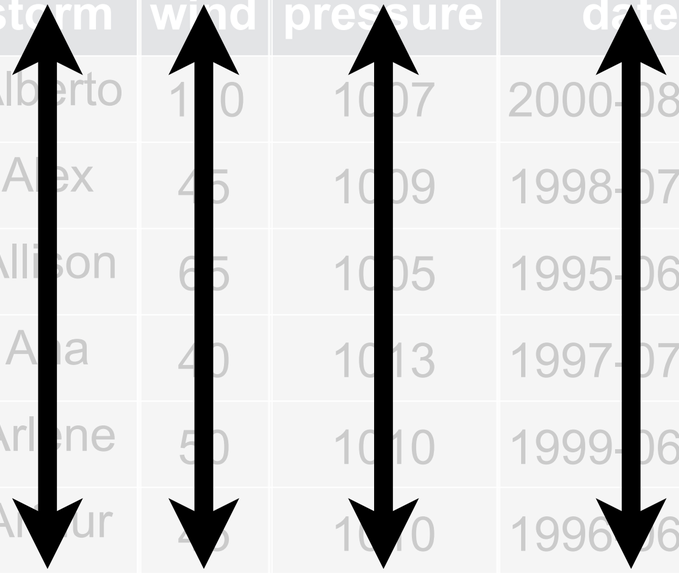
pollution

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

- Storm name
- Wind Speed (mph)
- Air Pressure

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



- Storm name
- Wind Speed (mph)
- Air Pressure
- Date

cases

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

pollution

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

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

- Storm name
- Wind Speed (mph)
- Air Pressure
- Date

cases

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

- Country

pollution

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

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

- Storm name
- Wind Speed (mph)
- Air Pressure
- Date

cases

Country			
FR	7000	6900	7000
DE	5800	6000	6200
UK	15000	14000	13000

- Country
- Year

pollution

city	particle size	amount (µg/m³)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

- Storm name
- Wind Speed (mph)
- Air Pressure
- Date

cases

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

- Country
- Year
- Count

pollution

city	particle size	amount (µg/m³)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

- Storm name
- Wind Speed (mph)
- Air Pressure
- Date

cases

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

- Country
- Year
- Count

pollution

city	particle size	amount (µg/m³)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

- City

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

- Storm name
- Wind Speed (mph)
- Air Pressure
- Date

cases

Country			
FR	7000	6900	7000
DE	5800	6000	6200
UK	15000	[illegible]	13000

- Country
- Year
- Count

pollution

city	particle size	amount (µg/m³)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

- City
- Amount of large particles

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

- Storm name
- Wind Speed (mph)
- Air Pressure
- Date

cases

Country			
FR	7000	6900	7000
DE	5800	6000	6200
UK	15000	14000	13000

- Country
- Year
- Count

pollution

city	particle size	amount (µg/m³)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

- City
- Amount of large particles
- Amount of small particles

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

```
storms$storm
storms$wind
storms$pressure
storms$date
```

cases

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

```
cases$country
names(cases)[-1]
unlist(cases[1:3, 2:4])
```

pollution

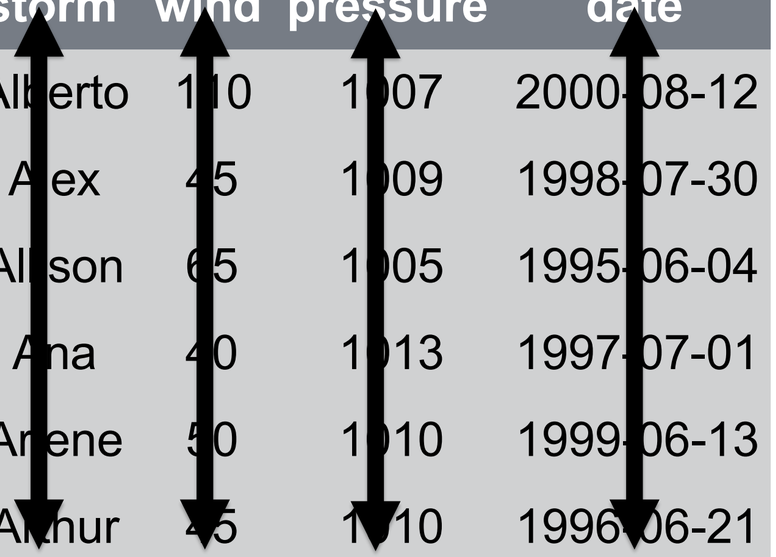
city	particle size	amount (µg/m³)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

```
pollution$city[1,3,5]
pollution$amount[1,3,5]
pollution$amount[2,4,6]
```

Tidy data

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



1 Each **variable** is saved in its own **column**.

$$\text{ratio} = \frac{\text{pressure}}{\text{wind}}$$

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

```
storms$pressure / storms$wind
```

```
950  /  110  →  8.6
1003 /  45   →  22.3
987  /  65   →  15.2
1004 /  40   →  25.1
1006 /  50   →  20.1
1000 /  45   →  22.2
```

Tidy data

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

1 Each **variable** is saved in its own **column**.

2 Each **observation** is saved in its own **row**.

Tidy data

storms

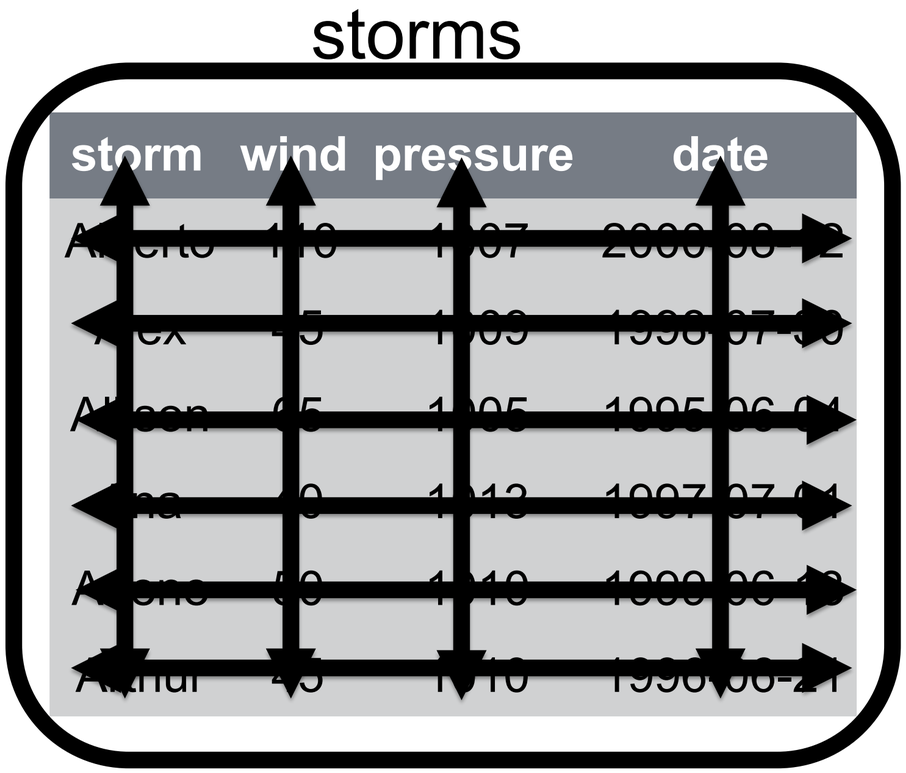
storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

1. Each **variable** is saved in its own **column**.

2. Each **observation** is saved in its own **row**.

3. Each "type" of observation stored in a **single table** (here, storms).

Tidy data



1 Each **variable** is saved in its own **column**.

2 Each **observation** is saved in its own **row**.

3 Each "type" of observation stored in a **single table** (here, storms).

Recap: Tidy data

123 Variables in columns, observations in rows, each type in a table

Easy to access variables

Automatically preserves observations

Are these tables tidy?

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

cases

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

pollution

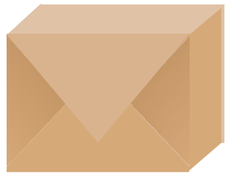
city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

Tidying Data

So, what do we do if the data breaks one (or more!) rule of tidy data?

Well, we bring it back into compliance!

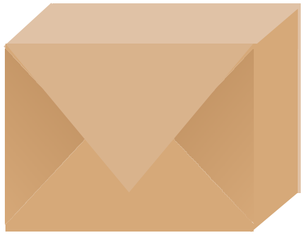
In order to tidy - or wrangle - the data, we will use the following functions from the **tidyr** package.



tidyr

tidyr

tidyr



A package that reshapes the layout of tables.

Four main functions: pivot_longer(), pivot_wider(), unite()

and **separate()**

```
# install.packages("tidyr")
library(tidyr)
? pivot_longer
? pivot_wider
? unite
? separate
```

How does pivoting work?

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

How does pivoting work?

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

Country	Year	n

How does pivoting work?

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

Country	Year	n
FR	2011	7000

How does pivoting work?

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

How does pivoting work?

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

How does pivoting work?

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

How does pivoting work?

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

How does pivoting work?

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

How does pivoting work?

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

How does pivoting work?

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

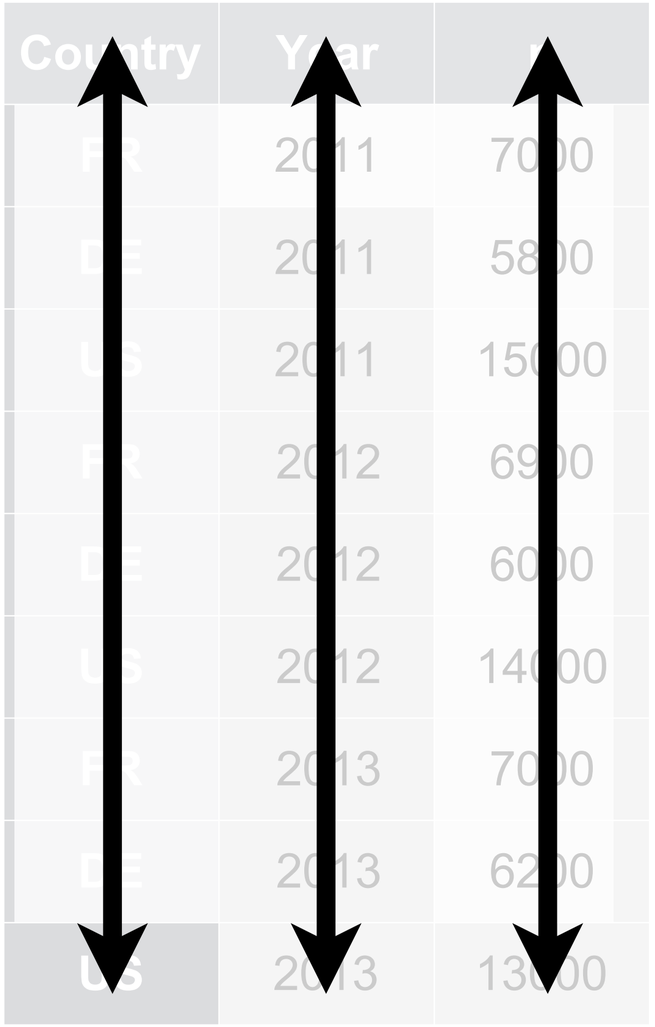
How does pivoting work?

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

How does pivoting work?

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



How does pivoting work?

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

How does pivoting work?

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

Pivot_longer()

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

How does pivoting work?

key (former column names)

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

How does pivoting work?

key **value** (former cells)

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

pivot_longer()

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

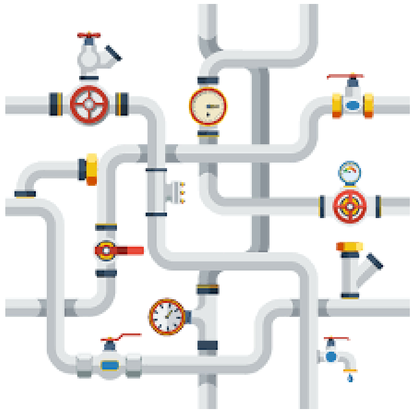
Collapses multiple columns into two columns:

1. a **key** column that contains the former column names
2. a **value** column that contains the former column cells

```
pivot_longer(
cases,
cols = 2:4,
names_to = "year",
values_to = "n")
```

```
cases %>%
  pivot_longer(
  cols = starts_with("20"),
  names_to = "year",
  values_to = "n")
```

```
Help! type
?pivot_longer()
```

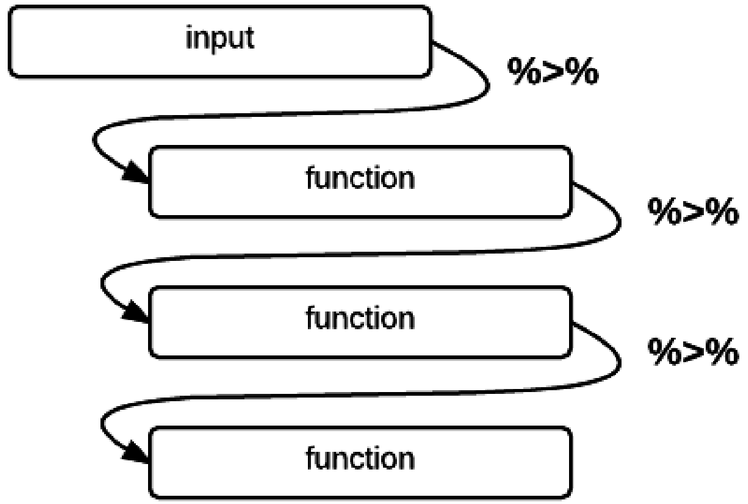


The pipe operator %>% or |>


```
library(dplyr)
pivot_longer(cases, cols =…)

cases %>% pivot_longer(cols =…

cases  %>% pivot_longer(_____   ,cols =…)
```



Shortcut to type %>%

Cmd + Shift + M (Mac)

Ctrl + Shift + M (Windows)

pivot_longer()

Collapses multiple columns into two columns:

1. a **key** column that contains the former column names
2. a **value** column that contains the former column cells

```
cases %>% pivot_longer(cols = 2:4, names_to = "year", values_to = "n")
```

data frame to reshape

pivot_longer()

Collapses multiple columns into two columns:

1. a **key** column that contains the former column names
2. a **value** column that contains the former column cells

```
cases %>% pivot_longer(cols = 2:4, names_to = "year", values_to = "n")
```

data frame to reshape

names or numeric indexes of columns to collapse

pivot_longer()

Collapses multiple columns into two columns:

1. a **key** column that contains the former column names
2. a **value** column that contains the former column cells

```
cases %>% pivot_longer(cols = 2:4, names_to = "year", values_to = "n")
```

data frame to reshape

names or numeric indexes of columns to collapse

name of the new key column (a character string)

pivot_longer()

Collapses multiple columns into two columns:

1. a **key** column that contains the former column names
2. a **value** column that contains the former column cells

```
cases %>% pivot_longer(cols = 2:4, names_to = "year", values_to = "n")
```

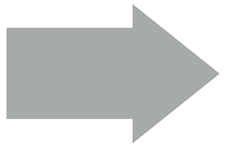
data frame to reshape

names or numeric indexes of columns to collapse

name of the new key column (a character string)

name of the new value column (a character string)

```
##     country  2011  2012  2013
## 1        FR  7000  6900  7000
## 2        DE  5800  6000  6200
## 3        US 15000 14000 13000
```



```
##     country year     n
## 1        FR 2011  7000
## 2        DE 2011  5800
## 3        US 2011 15000
## 4        FR 2012  6900
## 5        DE 2012  6000
## 6        US 2012 14000
## 7        FR 2013  7000
## 8        DE 2013  6200
## 9        US 2013 13000
```

```
cases %>% pivot_longer(cols = 2:4, names_to = "year", values_to = "n")
```

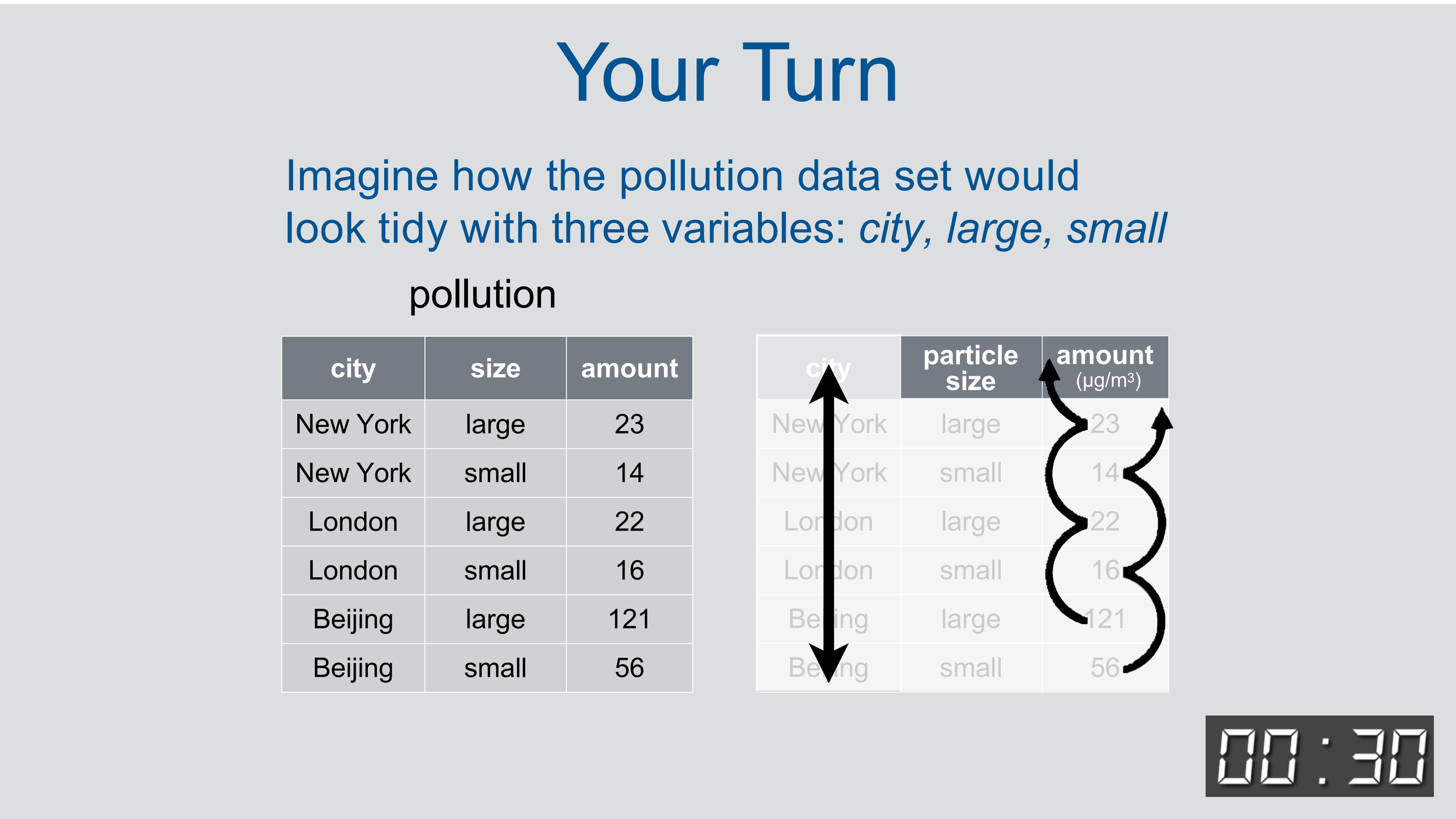
Your Turn

Imagine how the pollution data set would look tidy with three variables: *city, large, small*

pollution

city	size	amount
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/m³)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56



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

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

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	

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

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	

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

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	

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

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

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

key (new column names)

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

key**value** (new cells)

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

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

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

key (new column names)

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

key **value** (new cells)

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

pivot_wider()

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

pivot_wider()

Generates multiple columns from two columns:

1. each unique value in the **key** column becomes a column name
2. each value in the **value** column becomes a cell in the new columns

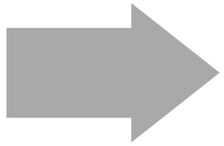
```
pollution %>% pivot_wider(names_from =  size, values_from =  amount)
```

data frame to reshape

column to use for keys (new columns names)

column to use for values (new column cells)

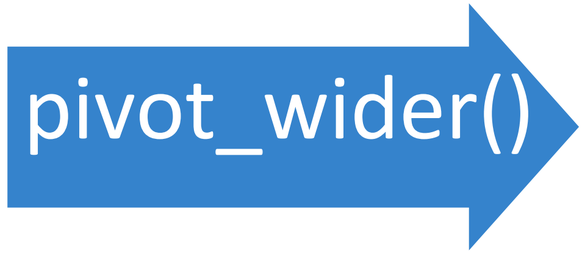
```
##       city  size amount
## 1 New York large     23
## 2 New York small     14
## 3   London large     22
## 4   London small     16
## 5  Beijing large    121
## 6  Beijing small     56
```



```
##       city large small
## 1  Beijing   121    56
## 2   London    22    16
## 3 New York    23    14
```

```
pollution %>% pivot_wider(names_from =  size, values_from =  amount)
```

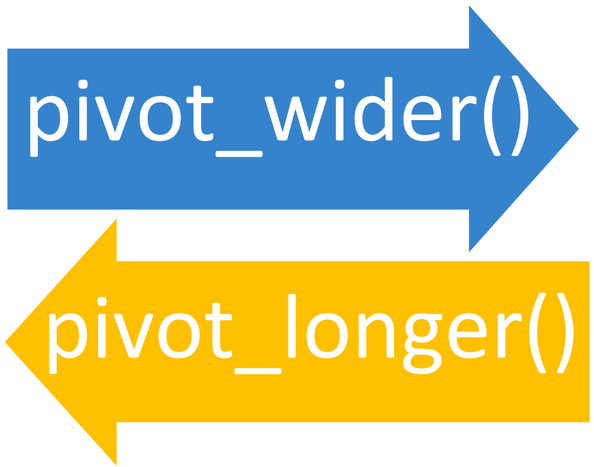
city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56



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

```
pollution %>% pivot_wider(names_from =  size, values_from =  amount)
```

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

```
pollution %>% pivot_wider(names_from =  size, values_from =  amount)
```

```
pollution %>% pivot_longer(names_to = "size", values_to = "amount",
cols = c(large,small))
```

separate() and unite()

There are three more variables hidden in storms:

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

- Year
- Month
- Day

separate()

splits a column by a character string separator.

```
storms %>% separate(date, c("year", "month", "day"), sep = "-")
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storms2

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

unite()

unites columns into a single column.

```
storms2 %>% unite("date", year, month, day, sep = "-")
```

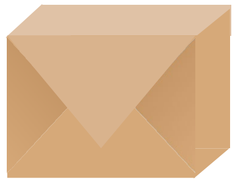
storms2

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

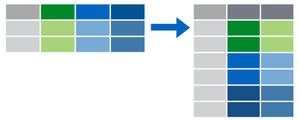
storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

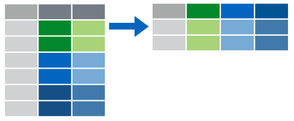
Recap: tidyr



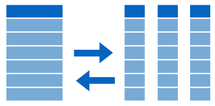
A package that reshapes the layout of data sets.



Make observations from variables with `pivot_longer()`



Make variables from observations with `pivot_wider()`



Split and merge columns with `unite()` and `separate()`

RStudio

Data tidying with tidyr : : CHEATSHEET

Tidy data is a way to organize tabular data in a consistent data structure across packages.
A table is tidy if:

Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own row

Access **variables** as **vectors**

Preserve **cases** in vectorized operations

Tibbles

AN ENHANCED DATA FRAME

Tibbles are a table format provided by the **tibble** package. They inherit the data frame class, but have improved behaviors:

- **Subset** a new tibble with], a vector with [[and $.
- **No partial matching** when subsetting columns.
- **Display** concise views of the data on one screen.

options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf**)** Control default display settings.

View() or **glimpse()** View the entire data set.

CONSTRUCT A TIBBLE

tibble(…**)** Construct by columns.

```
tibble(x = 1:3, y = c("a", "b", "c"))
```

tribble(…**)** Construct by rows.

```
tribble(~x,  ~y,
        1, "a",
        2, "b",
        3, "c")
```

Both make this tibble

```
A tibble: 3 × 2
      x     y
  <int> <chr>
1     1     a
2     2     b
3     3     c
```

Reshape Data - Pivot data to reorganize values into a new layout.

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

pivot_longer(data, cols, names_to = "name", values_to = "value", values_drop_na = FALSE**)**

"Lengthen" data by collapsing several columns into two. Column names move to a new names_to column and values to a new values_to column.

```
pivot_longer(table4a, cols = 2:3, names_to ="year",
  values_to = "cases")
```

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

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

pivot_wider(data, names_from = "name", values_from = "value"**)**

The inverse of pivot_longer(). "Widen" data by expanding two columns into several. One column provides the new column names, the other the values.

```
pivot_wider(table2, names_from = type,
  values_from = count)
```

Split Cells - Use these functions to split or combine cells into individual, isolated values.

table5

country	century	year
A	19	99
A	20	00
B	19	99
B	20	00

country	year
A	1999
A	2000
B	1999
B	2000

unite(data, col, …, sep = "_", remove = TRUE, na.rm = FALSE**)** Collapse cells across several columns into a single column.

```
unite(table5, century, year, col = "year", sep = "")
```

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174

separate_wider_delim(data, cols, delim, ..., names = NULL, names_sep = NULL, names_repair = "check unique", too_few, too_many, cols_remove = TRUE**)** Separate each cell in a column into several columns. Also **separate_wider_regex()** and **separate_wider_position().**

```
separate(table3, rate, sep = "/",
```

Expand Tables

Create new combinations of variables or identify implicit missing values (combinations of variables not present in the data).

expand(data, …**)** Create a new tibble with all possible combinations of the values of the variables listed in … Drop other variables.

```
expand(mtcars, cyl, gear,
carb)
```

complete(data, …, fill = list()**)** Add missing possible combinations of values of variables listed in … Fill remaining variables with NA.

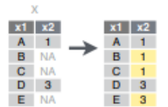
```
complete(mtcars, cyl, gear,
carb)
```

Handle Missing Values

Drop or replace explicit missing values (NA).

s(data, …**)** Drop rows containing NA's in … columns.

```
drop_na(x, x2)
```



fill(data, …, .direction = "down"**)** Fill in NA's in … columns using the next or previous value.

```
fill(x, x2)
```

https://rstudio.github.io/cheatsheets/tidyr.pdf

Acknowledgement

Garrett

PPT & PDF: https://github.com/rstudio/webinars/tree/master/05-Data-Wrangling-with-R-and-RStudio
Video: https://posit.co/resources/videos/data-wrangling-with-r-and-rstudio/

Thank you
&
questions?

Now let's practice in rstudio
(lab excercise)