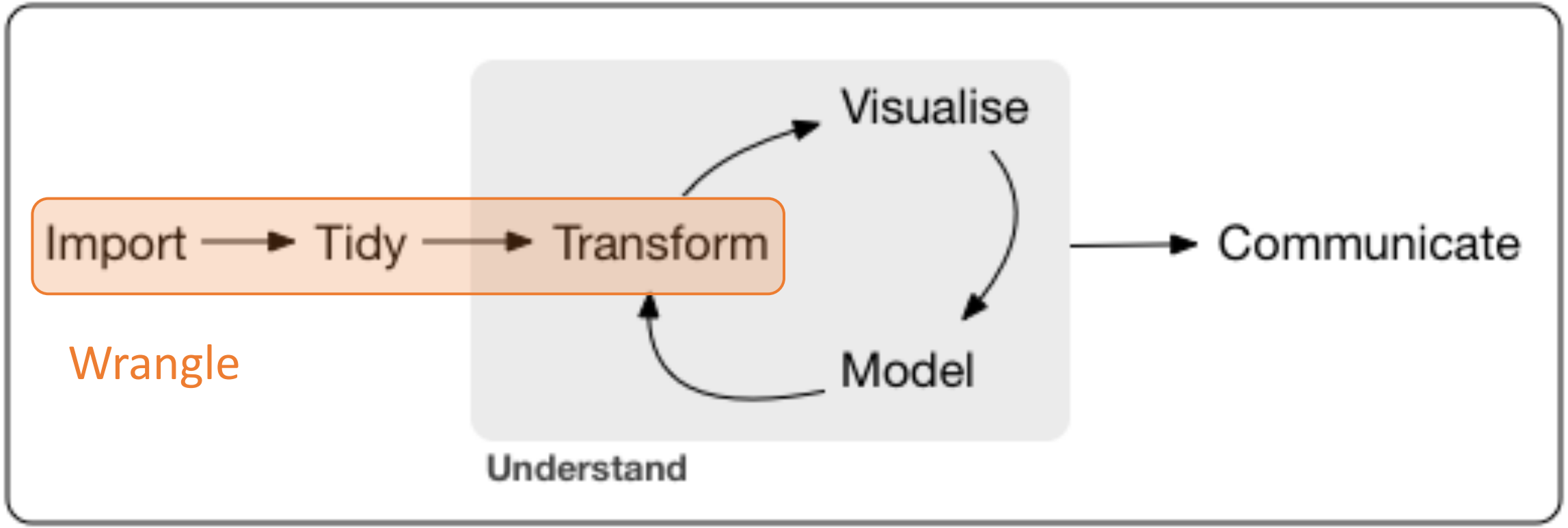
Week 4: Data wrangling (I)

A typical data science project :



🔥🔥**Kareem Carr** 🔥🔥 @kareem_carr · Jun 27, 2021

“on your left”

Tibbles

- The tibble package, part of the core tidyverse
- Tibbles are data frames, but they tweak some older behaviors to make life a little easier.
- There are two main differences in the usage of a tibble vs. a classic data.frame: printing and subsetting.
  - Printing
  - Subsetting

Import data into R

- Download “Sinai_covid.csv” from Blackoard and save it in your course folder.

Most of readr’s functions are concerned with turning flat files into data frames: **`read_csv`**

Writing to a file

readr also comes with functions for writing data back to disk: write_csv()

Tidy data

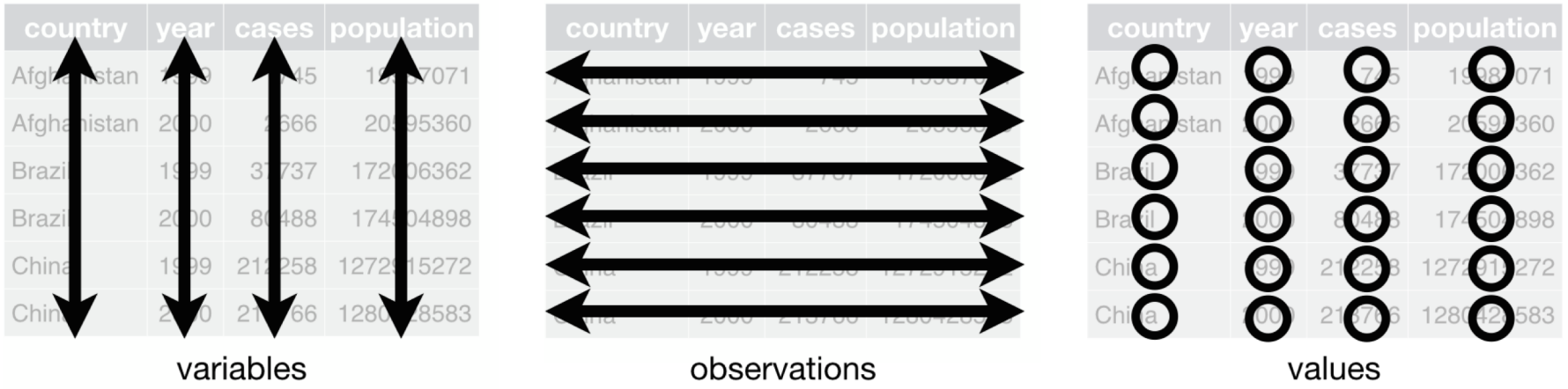
“Happy families are all alike; every unhappy family is unhappy in its own way.” — Leo Tolstoy

“Tidy datasets are all alike, but every messy dataset is messy in its own way.” — Hadley Wickham

We will be using the pipe **%>%** - it takes the output of one statement and makes it the input of the next statement. When describing it, you can think of it as a "THEN".

What makes a dataset tidy?

- Each variable must have its own column.
- Each observation must have its own row.
- Each value must have its own cell.



dplyr, ggplot2, and all the other packages in the tidyverse are designed to work with tidy data.

Pivoting

- Most data that you will encounter will be untidy.
- For most real analyses; you'll need to do some tidying.

Step 1: figure out what the variables and observations are (Sometimes you'll need to consult with the people who originally generated the data).

Step 2: resolve one of two common problems:

  - One variable might be spread across multiple columns.
  - One observation might be scattered across multiple rows.

- Typically, a dataset will only suffer from one of these problems; To fix these problems, you'll need two important functions in tidyr:

  pivot_longer()
  pivot_wider()

pivot_longer()

- A common problem is a dataset where some of the column names are not names of variables, but *values* of a variable.
- Take table4a: the column names 1999 and 2000 represent values of the year variable, and each row represents two observations.
- To tidy a dataset like this, we need to **pivot** the offending columns into a new pair of variables.
  1. The set of columns whose names are values (columns 1999 and 2000).
  2. The name of the variable to move the column names to (year).
  3. The name of the variable to move the column values to (cases).

pivot_longer() - Exercise

- How will you pivot_longer() table4b?

pivot_Wider()

- pivot_wider() is the opposite of pivot_longer().
- Take table2: an observation is a country in a year, but each observation is spread across two rows.
- To tidy this up, we only need two parameters:
  1. The column to take variable names from (type).
  2. The column to take values from (count).

Separating

- table3 has a different problem: we have one column (rate) that contains two variables (cases and population).
- To fix this problem, we'll need the separate() function.

Case Study

- The tidyr::who dataset contains tuberculosis (TB) cases broken down by year, country, age, gender, and diagnosis method.
- This is a very typical real-life example dataset.
- The best place to start is to gather the columns that are not variables.

Can you identify them?

Case Study

1. country, iso2, and iso3 are three variables that redundantly specify the country.
2. year is a variable.
3. new_sp_m014 - new_rel_f65 are counts of new TB cases recorded by group. Column names encode three variables that describe the group - these are values, not variables.
4. We need to gather all the columns (group). We know the cells represent the count of cases, so we'll use the variable cases.
5. There are a lot of missing values in the current representation, so we'll use values_drop_na

Case Study – data dictionary

- The first three letters of each column denote whether the column contains new or old cases of TB. In this dataset, each column contains new cases.
- The next two letters describe the type of TB:
  - rel stands for cases of relapse
  - ep stands for cases of extrapulmonary TB
  - sn stands for cases of pulmonary TB that could not be diagnosed by a pulmonary smear (smear negative)
  - sp stands for cases of pulmonary TB that could be diagnosed by a pulmonary smear (smear positive)
- The sixth letter gives the sex of TB patients. The dataset groups cases by males (m) and females (f).
- The remaining numbers gives the age group. The dataset groups cases into seven age groups:
  - 014 = 0 – 14 years old
  - 1524 = 15 – 24 years old
  - 2534 = 25 – 34 years old
  - 3544 = 35 – 44 years old
  - 4554 = 45 – 54 years old
  - 5564 = 55 – 64 years old
  - 65 = 65 or older

- What do you think should be done to tidy this data?

Core functions

Function	Utility	Package
%>%	"pipe" (pass) data from one function to the next	**magrittr**
mutate()	create, transform, and re-define columns	**dplyr**
select()	keep, remove, select, or re-name columns	**dplyr**
rename()	rename columns	**dplyr**
clean_names()	standardize the syntax of column names	**janitor**
as.character(), as.numeric(), as.Date(), etc.	convert the class of a column	**base** R
across()	transform multiple columns at one time	**dplyr**
filter()	keep certain rows	**dplyr**
distinct()	de-duplicate rows	**dplyr**
rowwise()	operations by/within each row	**dplyr**
add_row()	add rows manually	**tibble**
arrange()	sort rows	**dplyr**
recode()	re-code values in a column	**dplyr**

Column names - Automatic cleaning

- The function clean_names() from the package **janitor** standardizes column names
  1. Converts all names to consist of only underscores, numbers, and letters

  2. You can specify specific name replacements by providing a vector to the replace = argument (e.g. replace = c(onset = "date_of_onset"))

Column names - Manual name cleaning

- Re-naming columns manually is often necessary
- Re-naming is performed using the rename() function from the dplyr package, as part of a pipe chain.
- rename() uses the style NEW = OLD (the new column name is given before the old column name)

Select columns

- Use select() from **dplyr** to select, specify the order, and remove columns

Column creation and transformation

- Use the function mutate() to add a new column, or to modify an existing one.
- The syntax is: mutate(new_column_name = value or transformation)