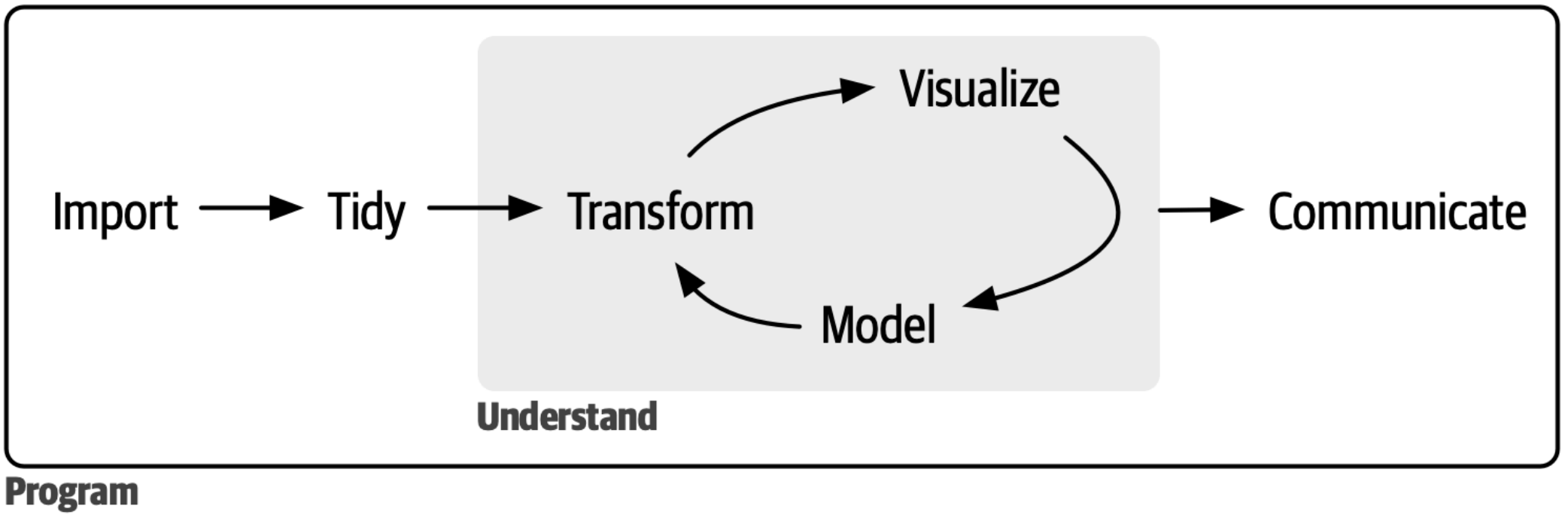
Statistical Computing with R

Part 7: Transform, Tidy and Communicate

Birgit Erni
Department of Statistical Sciences, University of Cape Twon

2024-02-26

Data Science



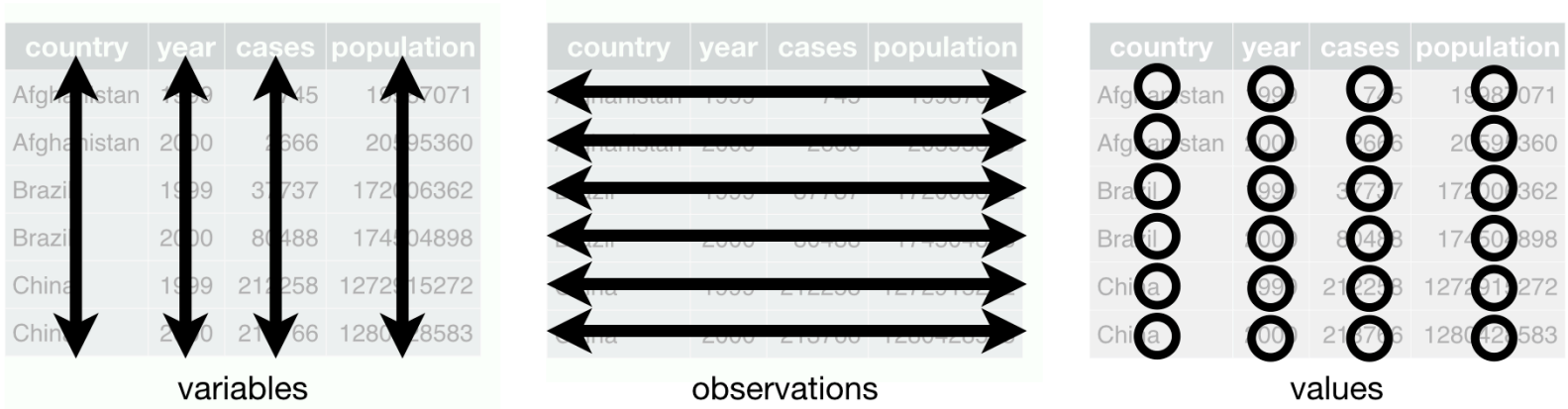
R for Data Science https://r4ds.hadley.nz/

- Statistical Computing in Data Science

Today: Tidy, Transform, Communicate

- Quarto (communicate)
- `dplyr` and `tidyverse` (tidy and transform)
- reshaping and merging data frames (import, tidy, transform)
- strings and regular expressions
- dates

Tidy Data



- observations in rows
- variables in columns
- values in cells

tidyverse: suit of packages for tidy data

`dplyr`

- d for data frame
- ply for '..ply' type functions (e.g. apply, tapply, ...)
- r for R
- pronounced: 'dee-ply-r'

`dplyr` is part of the *tidyverse*, a group of packages that helps to work with tidy data, and provides high-level functions that can deal with such data in a consistent way.

`dplyr` works with *verbs* to manipulate data frames:

Example

- flights that departed New York City in 2013 (`flights` in library `nycflights13`)

```
1 library(nycflights13)
2 library(dplyr)
3
4 dim(flights)
```

tibble is a special kind of data frame, better for large data sets

Example

```
[1] 336776     19
```

```
1 head(flights)
```

```
# A tibble: 6 × 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
1  2013     1     1      517            515         2      830            819
2  2013     1     1      533            529         4      850            830
3  2013     1     1      542            540         2      923            850
4  2013     1     1      544            545        -1     1004           1022
5  2013     1     1      554            600        -6      812            837
6  2013     1     1      554            558        -4      740            728
# ℹ 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
1 names(flights)
```

```
 [1] "year"           "month"          "day"            "dep_time"
 [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
 [9] "arr_delay"      "carrier"        "flight"         "tailnum"
[13] "origin"         "dest"           "air_time"       "distance"
[17] "hour"           "minute"         "time_hour"
```

```
1 View(flights)
```

Operations on Rows

- `arrange` – sort rows desc(dep_times) distinct
- `distinct`
- `filter`

filter(): subset rows

- select rows that meet condition

```
filter(flights, month == 1, day == 1)  # January 1st
filter(flights, month == 1 | month == 2)  # Jan or Feb
```

Exercise: Select all flights with departure delay of more than 30 minutes.

Select distinct rows

- find all distinct combinations of origin-destination

```
1 distinct(flights, origin, dest)
```

Operations on Columns

- mutate
- select
- rename
- relocate

`select()`

- select columns / variables.

```
select(flights, distance, arr_delay)
select(flights, !year:day)
```

Exercise: Find the flight with the longest (maximum) departure delay.

Rename variables: rename()

Rename variable `arr_delay` to `new.arr.delay`

```
1  rename(flights, new.arr.delay = arr_delay)
2  head(flights)
```

Exercise: Rename the variable as in the original data frame.

Add new columns: `mutate()

- define / calculate new variables (added as columns to the data frame), usually derived from other columns / variables.

```
mutate(flights,
  gain = arr_delay - dep_delay,
  speed = distance / air_time * 60)
```

Randomly sample rows

- `sample_n()` and `sample_frac()`: choose a random sample of n rows, or select a proportion of rows randomly from the original data set.
- useful for bootstrapping or cross-validation

```
sample_n(flights, 10, replace = TRUE)
sample_frac(flights, 0.01)
```

summarise()

- calculate summary statistics (several) for columns in the data frame

```
1  summarise(flights, mean(dep_delay), mean(arr_delay))
```

Exercise: Fix the function so that it does not return NAs.

Operations on Groups (grouped rows)

- `group_by()`
- `summarize`
- splits data (rows) into groups, the second argument is a grouping (factor) variable
- often followed by `summarise`, which now applies the function to every group.

```
1 dest <- group_by(flights, dest)
2 summarise(dest, mean(arr_delay, na.rm = TRUE),
3           dep.delay = mean(dep_delay))
```

Operations on Groups (grouped rows)

```
# A tibble: 105 × 3
   dest  `mean(arr_delay, na.rm = TRUE)` dep.delay
   <chr>                             <dbl>     <dbl>
 1 ABQ                                4.38     13.7
 2 ACK                                4.85      6.46
 3 ALB                               14.4      NA
 4 ANC                               -2.5      12.9
 5 ATL                               11.3      NA
 6 AUS                                6.02     NA
 7 AVL                                8.00     NA
 8 BDL                                7.05     NA
 9 BGR                                8.03     NA
10 BHM                               16.9      NA
# ℹ 95 more rows
```

Pipe operator `\>`

- takes object on the left-hand-side and pipes it into the function call on the right-hand-side – literally, drops it in as the first argument.

```
1  flights |> head(10)
```

Exercise: Use the pipe operator to select only rows with origin JFK, create a variable speed = distance / air_time, and calculate average speed.

`x |> f(y)` is equivalent to `f(x, y)`

`x |> f(y) |> g(z)` is equivalent to `g(f(x, y), z)`

like a `then`

```r
flights |>
  filter(dest == "IAH") |>
  group_by(year, month, day) |>
  summarize(arr_delay = mean(arr_delay, na.rm = TRUE))
```

Prac: `dplyr`

Use the `dplyr` functions on the `gapminder` data to:

1. select cases with life expectancy > 60
2. select cases from South Africa, with year > 1980, and calculate average life expectancy, and average GDP per capita income
3. repeat the above with the pipe operator
4. group by country, and calculate maximum, minimum, mean life expectancy, and n (number of observations/years)
5. extract life expectancy in 2007 for all countries
6. 2007 average life expectancy and GDP per capita per continent

Reshape Data Frames

Ideally, we want *each variable as a column, each observation in a row*. This is the form of **tidy data**, and many R functions expect tidy data, mainly because R works best with vectors = variables.

Messy data comes in many forms.

Too wide:

- column names are actually values
- e.g. year as column name
- year is a variable

```
1 library(tidyr)
2 table4a
```

Too long:

- values for one observation in multiple rows
- variable names as entries

```
1 table2
```

Reshape: Wide to Long

`melt`: To obtain a longer data frame we can `melt` the wide data frame. The `melt` function is from library `reshape2`.

Example: Under-5 mortality rate. Year is a variable, and its values should be in a column with names 'year'.

```
library(reshape2)

mort <- read.csv("unicef-u5mr.csv")
head(mort)    # wide format

mort.melt <- melt(mort, id.vars = "CountryName")
mort.melt
```

Better. Now it is much easier to select rows by year.

Reshape: Long to Wide

To make data frames wider, we can use the `dcast` function. `dcast` has a `formula` argument. For example, if we want to put the mortality data back into wide format, the formula should specify that we want `CountryName` as rows (x variables), and everything else as columns (y variables), except for `value`, which contains the values.

```
mort.wide <- dcast(mort.melt, formula = CountryName ~ variable)
mort.wide
```

Exercise:

Reformat `table2` into wider format such that it has one row per country and year. (`table2` from library `tidyr`.)

```
table2
```

Prac: Reshaping Data

Merging Data Frames

- merge several data files into a single data frame
- collect all variables for a specific person (or country, site, etc.)
- row variable: `id.variable` (used as link between different data files): `by` argument (or `by.x`, `by.y` if their names differ in the two data frames)
- data set 1: longitude and latitude per site, 2. temperature measurements for the sites.

```r
## data frame with site, longitude and latitude
sites <- data.frame(site = LETTERS[1:5],
                    lon = c(18, 25, 34, 20, 19),
                    lat = c(-27, -33, -29, -30, -31))
sites

## data frame with temperature measured at sites
temp <- data.frame(site = c("B", "C", "A", "A", "D"),
                   temp = c(25, 16, 40, 23, 19))
temp

merge(x = sites, y = temp, by = "site")
```

Prac: Merging Data Frames

1. Read in the CO2 and population size data sets (Vula). These are from Gapminder:
    - indicator gapminder population.xlsx
    - indicator CDIAC carbon_dioxide_emissions_per_capita.xlsx
2. Extract 2010 data from both.
3. Merge
4. Population size of country with highest per capita CO2 emissions in 2010? (1 765 513)
5. Number of rows in the merged data frame? (275)
6. What is the correlation between per capita CO2 emissions and population size? (-0.02)