Tidy Data

Nick Kuipers

Today's objectives

- Identify the main R data structures
- Understand the virtues of tidy data
- Gain familiarity with the tools of tidyverse
- Perform the workflow, moving from raw to tidy data

R Data Structures

1. *1-dimensional data*

- vectors (must be composed of single class)
- lists (may be multiple classes)

2. *2-dimensional data*

- matrices (must be composed of single class)
- data frames (may be multiple classes)

Elements of data

Data in R are principally composed of four types of elements: - character (i.e., strings) - numeric (i.e., 1.2, 1.3, 1.4. . .) - integer (i.e., 1, 2, 3, 4) - logical (i.e., Boolean)

Example	Type
“a”, “swc”	character
2, 15.5	numeric
2L (Must add a `L` at end to denote integer)	integer
`TRUE`, `FALSE`	logical

1-Dimensional Data: Vector properties

Upon encountering a vector, you can examine its properties to figure out what you might do with it.

- Class, `class()`, or what type of object it is (same as `type()` in Python).
- Length, `length()`, how many elements it contains (same as `len()` in Python).
- Attributes, `attributes()`, additional arbitrary metadata.

Recall that lists and vectors differ in the types of their elements: all elements of an atomic vector must be the same type, whereas the elements of a list can have different types.

1-Dimensional Data: Creating vectors

- Vectors are usually created with `c()`, which is short for concatenate:

```
atom_vec1 <- c(1, 2, 3) # Numeric

atom_vec2 <- c(1L, 2L, 3L) # Integer

atom_vec3 <- c("1", "2", "3") # Character

atom_vec4 <- c("1", 2, 3L) # ???
```

- Challenge: What do you think the class is of a heterogeneous vector?

```
class(atom_vec4)
```

1-Dimensional Data: Creating vectors

Vectors are usually created with `c()`, which is short for concatenate:

```
atom_vec1 <- c(1, 2, 3) # Numeric

atom_vec2 <- c(1L, 2L, 3L) # Integer

atom_vec3 <- c("1", "2", "3") # Character

atom_vec4 <- c("1", 2, 3L) # ???
```

- Challenge: What do you think the class is of a (coerced) vector?

```
class(atom_vec4)
```

```
## [1] "character"
```

1-Dimensional Data: List properties

Lists are similar to vectors but differ in that their composite elements can be of any type.

- Lists can contain other lists (nested lists/recursive listing)
- Lists can contain multiple vectors
- For most of you, you will encounter lists as the object returned from certain canned functions:
  - Regression models (lm())
  - Graphical objects (ggplot2())

1-Dimensional Data: Creating lists

Lists can be composed of multiple vectors of different types

```
vector_a = c("nick", "jae")
vector_b = c("kuipers", "kim")
vector_c = c(28, 34)

list(vector_a, vector_b, vector_c)
```

```
## [[1]]
## [1] "nick" "jae"
##
## [[2]]
## [1] "kuipers" "kim"
##
## [[3]]
## [1] 28 34
```

2-Dimensional Data: Matrices

- Matrices are composed of rows and columns, in which each element is the same type (usually numeric)
- Matrices generally do not contain meta-data (i.e., column names)

```
example_matrix = matrix(data = c(1, 2, 3, 4),
                        nrow = 2,
                        ncol = 2)
example_matrix
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

2-Dimensional Data: Data frames

- Data Frames are the **de facto** object in R. It is usually what we talk about when we talk about "data"
- Data frames are lists of vectors—vectors which may be different types of characters.

```
vector1 = c("king", "prince")
vector2 = c(34, 28)
vector3 = c("Jae", "Nick")

example_df = data.frame(vector1, vector2, vector3)
str(example_df)
```

```
## 'data.frame':    2 obs. of  3 variables:
##  $ vector1: chr  "king" "prince"
##  $ vector2: num  34 28
##  $ vector3: chr  "Jae" "Nick"
```

2-Dimensional Data: Data frames (cont'd)

- We often want to name our columns, which is easy in base R
- We also care about the *type* of variable that each column is assigned. Data frames often default to factors which is dangerous. But sometimes we like factors.

```
names(example_df) = c("status", "age", "name")
example_df$status = as.factor(example_df$status)
str(example_df)
```

```
## 'data.frame':    2 obs. of  3 variables:
##  $ status: Factor w/ 2 levels "king","prince": 1 2
##  $ age   : num  34 28
##  $ name  : chr  "Jae" "Nick"
```

2-Dimensional Data: Data frames (cont'd)

- Usually, we don't construct data frames from nothing. We have large datasets that we 'read in'' using canned functions. -read.csv(base R) -read.dta(foreign) -read.dta13(readstata13) -read.spss(foreign) -- N.B. be sure to incude argumentto.data.frame=T'

2-Dimensional Data: Thinking About the Unit of Analysis

- Most of the time, we are interested in seeing how variables correlate with one another, for which we need more than a single-dimensional data structure.
- How we approach the task of constructing 2-dimensional data structures has **extremely important** consequences for any resulting analyses.
- For instance, how should we structure our data for an analysis of:
  - Constituency-level democrat vote share, 1945-2020
  - Relationship between fathers and sons' heights, 1945-2020

Introducing Tidy Data and Entering the Tidyverse

- **Tidy data constitutes a set of principles regarding the proper structuring of data sets:** "Tidy data sets are easy to manipulate, model and visualize, and have a specific structure: each variable is a column, each observation is a row, and each type of observational unit is a table." - Hadley Wickham

1. Variables -> **Columns**
2. Observations -> **Rows**
3. Values -> **Cells**

Introducing Tidy Data and Entering the Tidyverse

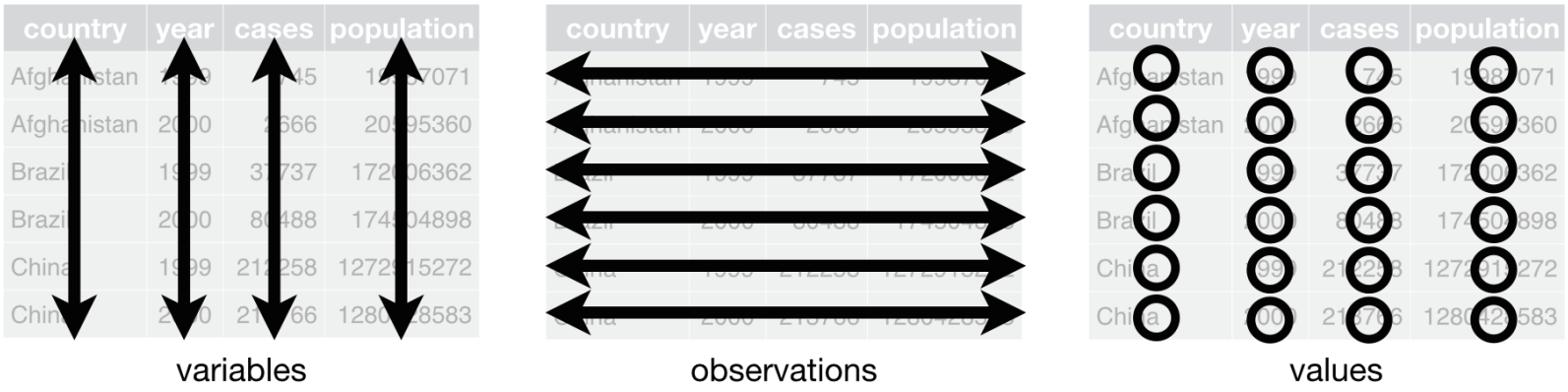

Figure 1: Tidy Data Example

If dataframes are tidy, it's easy to transform, visualize, model, and program them using tidyverse packages (a whole workflow).

Getting from untidy data to tidy data

Frequently, the data we encounter in the wild is untidy. There are many hallmarks of untidy data:

- Poorly labelled variables
- Variables are stored in rows, which leads to:
- Observations being repeated
- Values which could be separated are merged together
- General typos/filling/errors

Getting from untidy data to tidy data – Overview

Overall workflow:

1. Get structure of the data frame in place (i.e., make sure unit of analysis is correct)
2. Fix variable labels and select relevant columns
3. Negotiate missing values (are they intentional or unintentional?)
4. Split/merge variables
5. Clean up messy values/typos
6. Recode values

Getting from untidy data to tidy data – Tools

Overall workflow:

1. Get structure of the data frame in place (i.e., make sure unit of analysis is correct) (`pivot_longer` and `pivot_wider`)
2. Fix variable labels and select relevant columns (`select` and `rename`)
3. Negotiate missing values (are they intentional or unintentional?) (`filter`)
4. Split/merge variables (`separate`, `unite`, `mutate`)
5. Clean up messy values/typos (`mutate`)
6. Recode values (`mutate`, `case_when`)

Sidebar 1: pivot_longer and pivot_wider

Frequently, we want to reshape the raw data. The most common reshaping is from wide to long format. Here, we use `pivot_longer`:

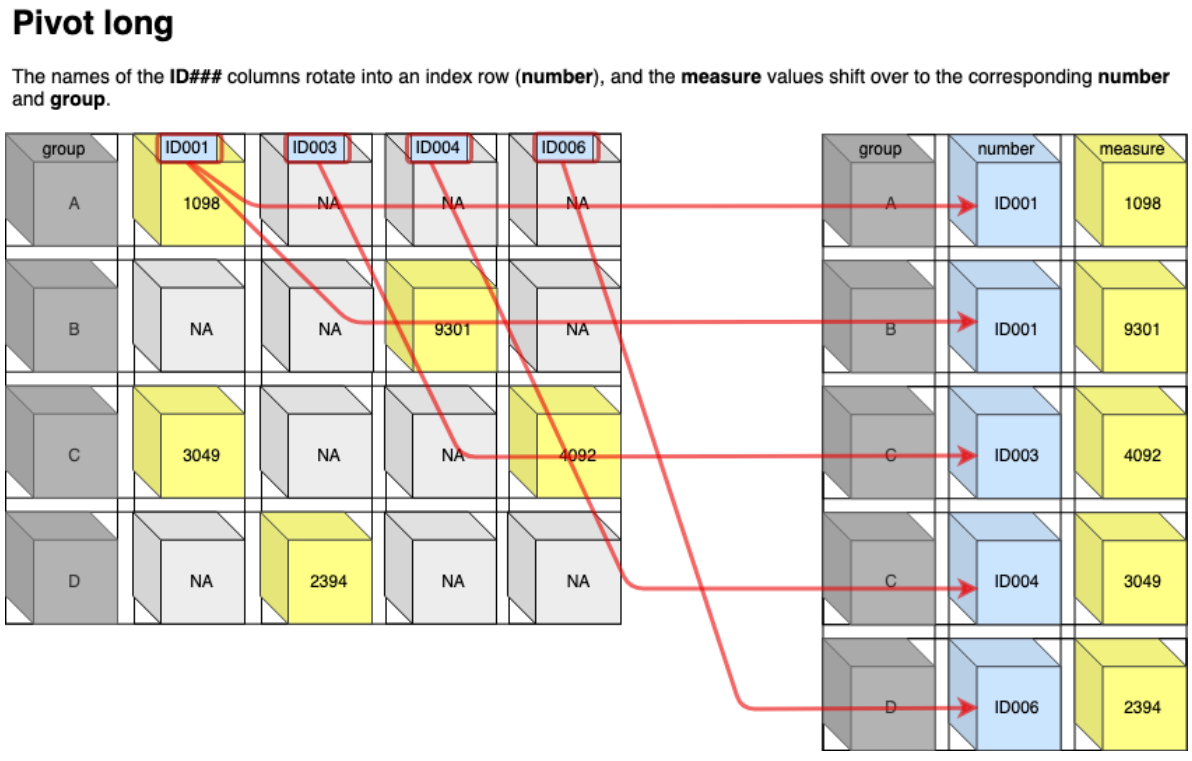

Figure 2: Pivot Data Longer

Sidebar 2: the pipe operator (overview)

- `R` is an object oriented programming language, which means that we perform transformations over an object—and then perform more transformations over that same object.
- This line-by-line transformation is occassionally cumbersome.
- Enter the **pipe operator** (%>%), which allows users to 'chain' together transformations. Think of the pipe operator as $f(g(x))$.

Sidebar 2: the pipe operator (example)

```r
df <- data.frame(col1 = c("a", "b", "c"),
                 col2 = c(1, 2, 3),
                 col3 = c(4, 5, 6))
#base R
row_a <- df[df$col1 == "a",]
sum_row_1 <- sum(row_a$col2, row_a$col3)
#chaining with dplyr
sum_row_2 <-
  df %>%
  filter(col1 == "a") %>%
  select(col2, col3) %>%
  sum()

sum_row_1
sum_row_2
```

```
## [1] 5
## [1] 5
```

An extended example: Uganda Council Elections (2011)

1. In the github repo, download the Ugandan elections data in the lecture_notes» week3 » data folder
   - `uganda_councillor_elections2011.csv`
2. Open up an instance of RStudio and follow along!!