Section 001 CDS 101 *Introduction to Computational and Data Sciences*

Class 11: Data wrangling IV

February 27, 2018

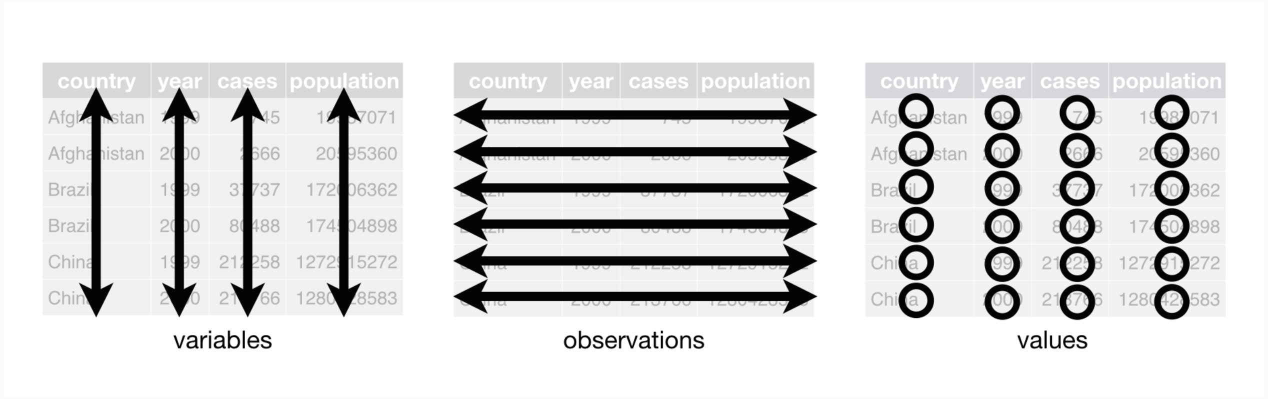
General

Annoucements

- Reading for next class: *R for Data Science*
  - From chapter 12: section 12.1 through to the end of section 12.3
- Homework 2 posted, due on Friday, March 9th by 11:59pm (Friday before Spring Break)

Tidy data

Principles



1. Each variable must have its own column.
2. Each observation (case) must have its own row.
3. Each value must have its own cell.

Why should we care?

First, according to *R for Data Science*,

Why should we care?

First, according to *R for Data Science*,

1. There's a general advantage to picking one consistent way of storing data. If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.

2. There's a specific advantage to placing variables in columns because it allows R's vectorised nature to shine. As you learned in mutate and summary functions, most built-in R functions work with vectors of values. That makes transforming tidy data feel particularly natural.

Why should we care?

First, according to *R for Data Science*,

1. There's a general advantage to picking one consistent way of storing data. If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.

2. There's a specific advantage to placing variables in columns because it allows R's vectorised nature to shine. As you learned in mutate and summary functions, most built-in R functions work with vectors of values. That makes transforming tidy data feel particularly natural.

Translation: *Getting data into this form allows you to work on entire columns at a time using short and memorable commands*

Why should we care?

First, according to *R for Data Science*,

1. There's a general advantage to picking one consistent way of storing data. If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.

2. There's a specific advantage to placing variables in columns because it allows R's vectorised nature to shine. As you learned in mutate and summary functions, most built-in R functions work with vectors of values. That makes transforming tidy data feel particularly natural.

> ***Translation:*** *Getting data into this form allows you to work on entire columns at a time using short and memorable commands*

If you've programmed before, you are probably familiar with loops. In other languages, data manipulation may require you to tell your computer to scan the tabular dataset **one cell at a time**.

Why should we care?

First, according to *R for Data Science*,

1. There's a general advantage to picking one consistent way of storing data. If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.

2. There's a specific advantage to placing variables in columns because it allows R's vectorised nature to shine. As you learned in mutate and summary functions, most built-in R functions work with vectors of values. That makes transforming tidy data feel particularly natural.

> ***Translation:*** *Getting data into this form allows you to work on entire columns at a time using short and memorable commands*

If you've programmed before, you are probably familiar with loops. In other languages, data manipulation may require you to tell your computer to scan the tabular dataset **one cell at a time**. R can do this,

Why should we care?

First, according to *R for Data Science*,

1. There’s a general advantage to picking one consistent way of storing data. If you have a consistent data structure, it’s easier to learn the tools that work with it because they have an underlying uniformity.

2. There’s a specific advantage to placing variables in columns because it allows R’s vectorised nature to shine. As you learned in mutate and summary functions, most built-in R functions work with vectors of values. That makes transforming tidy data feel particularly natural.

> ***Translation:*** *Getting data into this form allows you to work on entire columns at a time using short and memorable commands*

If you've programmed before, you are probably familiar with loops. In other languages, data manipulation may require you to tell your computer to scan the tabular dataset **one cell at a time**. R can do this, but it's slow...

Why should we care?

First, according to *R for Data Science*,

1. There's a general advantage to picking one consistent way of storing data. If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.

2. There's a specific advantage to placing variables in columns because it allows R's vectorised nature to shine. As you learned in mutate and summary functions, most built-in R functions work with vectors of values. That makes transforming tidy data feel particularly natural.

> ***Translation:*** *Getting data into this form allows you to work on entire columns at a time using short and memorable commands*

If you've programmed before, you are probably familiar with loops. In other languages, data manipulation may require you to tell your computer to scan the tabular dataset **one cell at a time**. R can do this, but it's slow...

The "vectorized" tools of `tidyverse` are both faster and easier to understand!

Why should we care?

- There's a theoretical foundation to this, actually

Why should we care?

- There's a theoretical foundation to this, actually
- Closely related to the formalism of *relational databases*

Why should we care?

- There's a theoretical foundation to this, actually
- Closely related to the formalism of *relational databases*
- If you follow these rules, your data will be in Codd's 3rd normal form

Why should we care?

- There's a theoretical foundation to this, actually
- Closely related to the formalism of *relational databases*
- If you follow these rules, your data will be in Codd's 3rd normal form (if this means anything to you)

Why should we care?

- There's a theoretical foundation to this, actually
- Closely related to the formalism of *relational databases*
- If you follow these rules, your data will be in Codd's 3rd normal form (if this means anything to you)
- Helpful if you are working with a large or complex enough dataset that you need to store in a formal database, such as SQL databases (Postgresql, Mysql)

Why should we care?

- There's a theoretical foundation to this, actually
- Closely related to the formalism of *relational databases*
- If you follow these rules, your data will be in Codd's 3rd normal form (if this means anything to you)
- Helpful if you are working with a large or complex enough dataset that you need to store in a formal database, such as SQL databases (Postgresql, Mysql)
- Practically speaking, the tidying process makes the categories in your data more clear

Why should we care?

- There's a theoretical foundation to this, actually
- Closely related to the formalism of *relational databases*
- If you follow these rules, your data will be in Codd's 3rd normal form (if this means anything to you)
- Helpful if you are working with a large or complex enough dataset that you need to store in a formal database, such as SQL databases (Postgresql, Mysql)
- Practically speaking, the tidying process makes the categories in your data more clear
- It makes analysis much easier too, because you can easily subdivide your data by category, and apply transformations where needed

Why should we care?

- There's a theoretical foundation to this, actually
- Closely related to the formalism of *relational databases*
- If you follow these rules, your data will be in Codd's 3rd normal form (if this means anything to you)
- Helpful if you are working with a large or complex enough dataset that you need to store in a formal database, such as SQL databases (Postgresql, Mysql)
- Practically speaking, the tidying process makes the categories in your data more clear
- It makes analysis much easier too, because you can easily subdivide your data by category, and apply transformations where needed
- Provides a standardized, "best practices" way to structure and store our datasets

Why should we care?

- There's a theoretical foundation to this, actually
- Closely related to the formalism of *relational databases*
- If you follow these rules, your data will be in Codd's 3rd normal form (if this means anything to you)
- Helpful if you are working with a large or complex enough dataset that you need to store in a formal database, such as SQL databases (Postgresql, Mysql)
- Practically speaking, the tidying process makes the categories in your data more clear
- It makes analysis much easier too, because you can easily subdivide your data by category, and apply transformations where needed
- Provides a standardized, "best practices" way to structure and store our datasets
    - Note that you may not collect or input your data straight into tidy format

Tidying ≠ Cleaning

- Data tidying does **not** encompass the entire data cleaning process
- Data tidying only refers to reshaping things, such as moving columns and rows around
- Cleaning operations, such as correcting spelling errors, renaming variables, etc., is a separate topic

`tidyr()` package

Summary of `tidyr()` package

Summary of `tidyr()` package

- Functions (commands) that allow you to reshape data

Summary of `tidyr()` package

- Functions (commands) that allow you to reshape data
- Oriented towards the kinds of datasets we've worked with previously, each column may be a different data type (numeric, string, logical, etc)

Summary of `tidyr()` package

- Functions (commands) that allow you to reshape data
- Oriented towards the kinds of datasets we've worked with previously, each column may be a different data type (numeric, string, logical, etc)
- Functions (commands) are typed in a way that's very similar to the `dplyr` *verbs*, such as `filter()` and `mutate()`

Summary of `tidyr()` package

- Functions (commands) that allow you to reshape data
- Oriented towards the kinds of datasets we've worked with previously, each column may be a different data type (numeric, string, logical, etc)
- Functions (commands) are typed in a way that's very similar to the `dplyr` *verbs*, such as `filter()` and `mutate()`
- `tidyr` verbs

Summary of `tidyr()` package

- Functions (commands) that allow you to reshape data
- Oriented towards the kinds of datasets we've worked with previously, each column may be a different data type (numeric, string, logical, etc)
- Functions (commands) are typed in a way that's very similar to the `dplyr` *verbs*, such as `filter()` and `mutate()`
- `tidyr` verbs
    - `gather()`: transforms wide data to narrow data

Summary of `tidyr()` package

- Functions (commands) that allow you to reshape data
- Oriented towards the kinds of datasets we've worked with previously, each column may be a different data type (numeric, string, logical, etc)
- Functions (commands) are typed in a way that's very similar to the `dplyr` *verbs*, such as `filter()` and `mutate()`
- `tidyr` verbs
    - `gather()`: transforms wide data to narrow data
    - `spread()`: transforms narrow data to wide data

Summary of `tidyr()` package

- Functions (commands) that allow you to reshape data
- Oriented towards the kinds of datasets we've worked with previously, each column may be a different data type (numeric, string, logical, etc)
- Functions (commands) are typed in a way that's very similar to the `dplyr` *verbs*, such as `filter()` and `mutate()`
- `tidyr` verbs
    - `gather()`: transforms wide data to narrow data
    - `spread()`: transforms narrow data to wide data
    - `separate()`: make multiple columns out of a single column

Summary of `tidyr()` package

- Functions (commands) that allow you to reshape data
- Oriented towards the kinds of datasets we've worked with previously, each column may be a different data type (numeric, string, logical, etc)
- Functions (commands) are typed in a way that's very similar to the `dplyr` *verbs*, such as `filter()` and `mutate()`
- `tidyr` verbs
    - `gather()`: transforms wide data to narrow data
    - `spread()`: transforms narrow data to wide data
    - `separate()`: make multiple columns out of a single column
    - `unite()`: make a single column out of multiple columns

Simple examples from textbook

Follow along in RStudio

Tidy gradebook dataset exercise

Download the Github Classroom repo linked in channel `#4-starters` on Slack and complete the following exercises:

1. Make the dataset tidy using either `gather()` or `spread()`. The tidy gradebook should have one observation per row, which is one grade per student per assignment.

2. Use the tidy gradebook and create a histogram that answers the question, "What was the grade distribution for the Midterm Exam?"

Remember to commit and push your work before leaving class!