Reshaping Data using `tidyr`

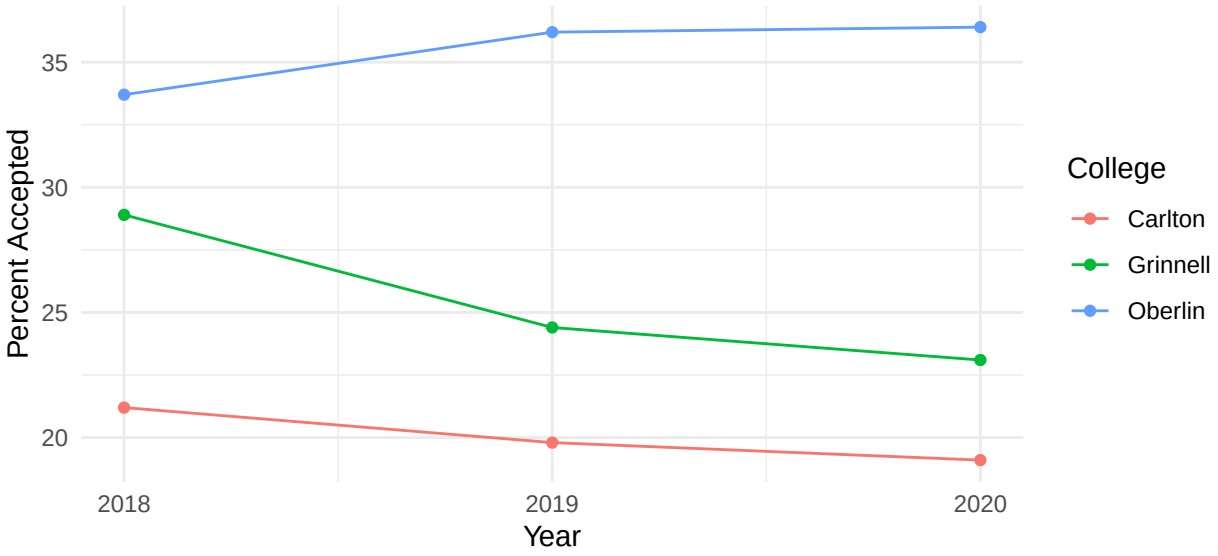
Contents

The tidyr package

This lab focuses on manipulating, cleaning, and preparing data for visualization (or other analyses) using packages from the tidyverse suite.

Motivation

Shown below are the acceptance rates over time of three different liberal arts colleges from 2018 to 2020:



When working with data, you must be able to connect the *format* of your data and a desired data visualization.

1) If creating an excel spreadsheet from this graphic, how do you think *most people* would record the data?
2) How should the data be formatted to match the conventions of `ggplot`?

Long vs. Wide Data

Data in a **wide format** record *many different values* or variables for a *single entity in a single row* (ie: acceptance rates for a college in different years, or different test scores for the same employee).

Data in a **long format** use *multiple rows* and a *single column* for the outcome or value of interest (ie: acceptance rate, test score, etc.) with additional columns identifying the meaning of that value.

Example:

Wide Format

EmployeeID	PreTest	PostTest
EP1619	55	60
EP1845	62	66
EP4321	88	100

Long Format

EmployeeID	TestTime	Score
EP1619	PreTest	55
EP1619	PostTest	60
EP1845	PreTest	62
EP1845	PostTest	66
EP4321	PreTest	88
EP4321	PostTest	100

The `ggplot2` package, as well as the implementations of many statistical models, expect data in **long format**. However, many data manipulations are easier to do in wide format; for example, calculating the average improvement from pre-test to post-test.

Tidy Data

Converting between "wide" and "long" formats is often the most challenging step in creating a "tidy" data set, or one that is fully prepared for graphing/modeling.

In general, **tidy data** are defined by the following criteria:

1. Every column is a variable 2. Every row is an observation 3. Every cell is a single value

This lab will introduce several data manipulation functions used to help tidy a data set into a more useful format.

Packages and Datasets

This lab primarily uses `tidyr` package, which is used to "tidy" or reshape data. It will also use the `ggplot2` package.

```
# Please install and load the following packages
# install.packages("tidyr")
library(tidyr)
library(ggplot2)
```

The lab will use several data sets in its examples:

```
collegeAdm = read.csv("https://raw.githubusercontent.com/grinnell-statistics/Grinnell_R_Tutorials/refs/
```

- **Description**: Admissions rates of three Midwestern liberal arts colleges according to acceptancer-ate.com

```
bluechips = read.csv("https://raw.githubusercontent.com/grinnell-statistics/Grinnell_R_Tutorials/refs/h
```

- **Description**: Closing prices on the first trading day of the year from 2010 to 2021 for four stocks that The Motley Fool calls "blue chip" investments.

```
polls <- read.csv("https://raw.githubusercontent.com/grinnell-statistics/Grinnell_R_Tutorials/refs/heads
```

- **Description**: Polling data leading up to the 2016 US Presidential Elections scraped from RealClearPolitics.com

Lab

At this point you will begin working with your partner. Please read through the text/examples and make sure you both understand before attempting to answer the embedded questions.

Pivoting between long and wide formats

Consider the `collegeAdm` data frame:

```
head(collegeAdm)
```

```
##   Adm_Rate Year  College
## 1     28.9 2018 Grinnell
## 2     24.4 2019 Grinnell
## 3     23.1 2020 Grinnell
## 4     21.2 2018  Carlton
## 5     19.8 2019  Carlton
## 6     19.1 2020  Carlton
```

These data are currently in "long" format, but we could convert them to a "wide" format using the `pivot_wider()` function:

```
## Pivot from long to wide to get 1 row per Year
wideCollegeAdm <- pivot_wider(collegeAdm,
                              id_cols = Year,
                              names_from = College,
                              values_from = Adm_Rate)
head(wideCollegeAdm)
```

```
## # A tibble: 3 x 4
##    Year Grinnell Carlton Oberlin
##   <int>    <dbl>   <dbl>   <dbl>
## 1  2018     28.9    21.2    33.7
## 2  2019     24.4    19.8    36.2
## 3  2020     23.1    19.1    36.4
```

The following arguments guide this transformation:

- `id_cols` determines what will be given its own row in the "wide" data set (ie: each row will be a unique value of the variable "Year")

- `names_from` defines the *single column* from the "long" data that should be *spread into multiple distinct columns* in the "wide" data (ie: each value of "College" is given a column named after it)
- `values_from` defines the *single column* from the "long" data containing the *values used to populate the cells* of the "wide" data (ie: the columns created for each "College" will contain the values of "Adm_Rate")

Notice what happens when `id_cols` and `names_from` are swapped:

```
## Pivot from long to wide to get 1 row per College
wideCollegeAdm2 <- pivot_wider(collegeAdm,
                               id_cols = College,
                               names_from = Year,
                               values_from = Adm_Rate)
head(wideCollegeAdm2)
```

```
## # A tibble: 3 x 4
##   College  `2018` `2019` `2020`
##   <chr>     <dbl>  <dbl>  <dbl>
## 1 Grinnell   28.9   24.4   23.1
## 2 Carlton    21.2   19.8   19.1
## 3 Oberlin    33.7   36.2   36.4
```

Similarly, the `pivot_longer()` function will transform "wide" data into "long" data:

```
pivot_longer(wideCollegeAdm2,
             cols = !College,
             names_to = "Year",
             values_to = "Adm_Rate")
```

```
## # A tibble: 9 x 3
##   College  Year  Adm_Rate
##   <chr>    <chr>    <dbl>
## 1 Grinnell 2018      28.9
## 2 Grinnell 2019      24.4
## 3 Grinnell 2020      23.1
## 4 Carlton  2018      21.2
## 5 Carlton  2019      19.8
## 6 Carlton  2020      19.1
## 7 Oberlin  2018      33.7
## 8 Oberlin  2019      36.2
## 9 Oberlin  2020      36.4
```

- `cols` defines the column(s) used in the pivot (`!College` will include everything but the variable "College"). The values of these variables will be collapsed into a single column.
- `names_to` is the name of the single column in the "long" data frame that will store the *column names* of the "wide" data frame
- `values_to` is the name of the single column in the "long" data frame that will store the *values* from the cells of the "wide" data frame

Note we could interchangeably use the argument `cols = c("2018", "2019", "2020")` or `cols = 2:4` (instead of `cols = !College`) to achieve the exact same result. The former approach explicitly names the columns that should be pivoted, and the later gives their index positions.

Question #1: Convert the `bluechips` data to a long format where each stock's closing price on the first trading day of each year is recorded in a single column named "Price".

Question #2: Starting with the long format data frame you created in Question #1, recreate the original `bluechips` data set using `pivot_wider()`.

Other `tidyr` functions

Pivoting or reshaping is often only one of many steps needed to tidy a data set. Another common occurrence is that data will contain multiple variables in a single column. For example, consider the "Date" and "Sample" columns in the `polls` data set:

```
head(polls)
```

```
##                     Poll        Date  Sample MoE Clinton..D. Trump..R.
## 1              Monmouth 7/14 - 7/16  688 LV 3.7          45        43
## 2               CNN/ORC 7/13 - 7/16  872 RV 3.5          42        37
## 3    ABC News/Wash Post 7/11 - 7/14  816 RV 4.0          42        38
## 4 NBC News/Wall St. Jrnl  7/9 - 7/13 1000 RV 3.1          41        35
## 5      Economist/YouGov  7/9 - 7/11  932 RV 4.5          40        37
## 6  Associated Press-GfK  7/7 - 7/11  837 RV  NA          40        36
##   Johnson..L. Stein..G.
## 1           5         1
## 2          13         5
## 3           8         5
## 4          11         6
## 5           5         2
## 6           6         2
```

The column "Date" contains two distinct variables, the start and end of the poll's sampling period. Similarly, "Sample" also contains two variables, the number of participants in the poll and the population that was sampled (registered voters or likely voters).

The `separate()` function is used to split a column into multiple new columns using a defined separator:

```
## Example #1
tidy_polls <- separate(polls,
                       col = Date,
                       into = c("Begin", "End"),
                       sep = " - ")
head(tidy_polls)
```

```
##                     Poll Begin  End  Sample MoE Clinton..D. Trump..R.
## 1              Monmouth  7/14 7/16  688 LV 3.7          45        43
## 2               CNN/ORC  7/13 7/16  872 RV 3.5          42        37
## 3    ABC News/Wash Post  7/11 7/14  816 RV 4.0          42        38
## 4 NBC News/Wall St. Jrnl   7/9 7/13 1000 RV 3.1          41        35
## 5      Economist/YouGov   7/9 7/11  932 RV 4.5          40        37
## 6  Associated Press-GfK   7/7 7/11  837 RV  NA          40        36
##   Johnson..L. Stein..G.
## 1           5         1
## 2          13         5
## 3           8         5
```

```
## 4          11         6
## 5           5         2
## 6           6         2
```

- `col` is the single column to be separated
- `into` indicates the names of the new columns produced by the separation
- `sep` is the character string used to determine how to split. In this example, the split happens when `-` surrounded by a space on each side is present.

In Example #2 (shown below), the "sep" argument is not explicitly given. In this situation, the default behavior of `separate()` is to try and guess an appropriate separator.

```
## Example #2
tidy_polls <- separate(polls,
                       col = Sample,
                       into = c("Size", "Population"))
head(tidy_polls)
```

```
##                     Poll        Date Size Population MoE Clinton..D. Trump..R.
## 1               Monmouth 7/14 - 7/16  688         LV 3.7          45        43
## 2                CNN/ORC 7/13 - 7/16  872         RV 3.5          42        37
## 3     ABC News/Wash Post 7/11 - 7/14  816         RV 4.0          42        38
## 4 NBC News/Wall St. Jrnl  7/9 - 7/13 1000         RV 3.1          41        35
## 5       Economist/YouGov  7/9 - 7/11  932         RV 4.5          40        37
## 6   Associated Press-GfK  7/7 - 7/11  837         RV  NA          40        36
##   Johnson..L. Stein..G.
## 1           5         1
## 2          13         5
## 3           8         5
## 4          11         6
## 5           5         2
## 6           6         2
```

While this is not generally recommended, it can work well if there's a clear pattern in your variable. More complex strings might require the use of *regular expressions*, a topic we'll cover later this semester.

Question #3 (Part A): Using either the `pivot_longer()` or `pivot_wider()` function, create a version of the "tidy_polls" data containing the variables "Candidate" and "Percentage", where "Candidate" is taken from the names of the last four columns of the data frame, and "Percentage" is taken from the values contained in these columns.

Question #3 (Part B): Using the `separate()` function, split the column "Candidate" (created in Part A) into two distinct columns containing the name of the candidate (ie: Clinton, Trump, etc.) and their political party (ie: D, R, etc.). *Hint*: periods, or `.`, are a special character in `R`, but you can reference one using the expression: `[.]`. You can also try letting `separate()` guess the proper splitting characters.

Practice

Question #4: The "airlines" data set (loaded below) contains data used in the article Should Travelers Avoid Flying Airlines That Have Had Crashes in the Past? that appeared on fivethirtyeight.com.

```
airlines <- read.csv("https://raw.githubusercontent.com/ds4stats/r-tutorials/master/tidying-data/data/ai
head(airlines)
```

```
##                 airline avail_seat_km_per_week incidents.1985_1999
## 1            Aer Lingus              320906734                   2
## 2             Aeroflot*             1197672318                  76
## 3 Aerolineas Argentinas              385803648                   6
## 4           Aeromexico*              596871813                   3
## 5            Air Canada             1865253802                   2
## 6            Air France             3004002661                  14
##   fatal_accidents.1985_1999 fatalities.1985_1999 incidents.2000_2014
## 1                         0                    0                   0
## 2                        14                  128                   6
## 3                         0                    0                   1
## 4                         1                   64                   5
## 5                         0                    0                   2
## 6                         4                   79                   6
##   fatal_accidents.2000_2014 fatalities.2000_2014
## 1                         0                    0
## 2                         1                   88
## 3                         0                    0
## 4                         0                    0
## 5                         0                    0
## 6                         2                  337
```

Recall that a "tidy" version of these data should satisfy the following:

- Each row is a single airline in a specific time period (ie: Air Canada in 1985-1999 or Alaska Airlines in 2000-2014)
- Each column contains only a single variable
- Each cell contains only a single value

Part A: Use `pivot_longer()` to gather the last six columns of the "airlines" data into a column named "accidents" and a column named "count".

Part B: Use `separate()` to split the "accidents" column into two variables named "var" and "years". *Hint*: remember that the period is a special character in `R`.

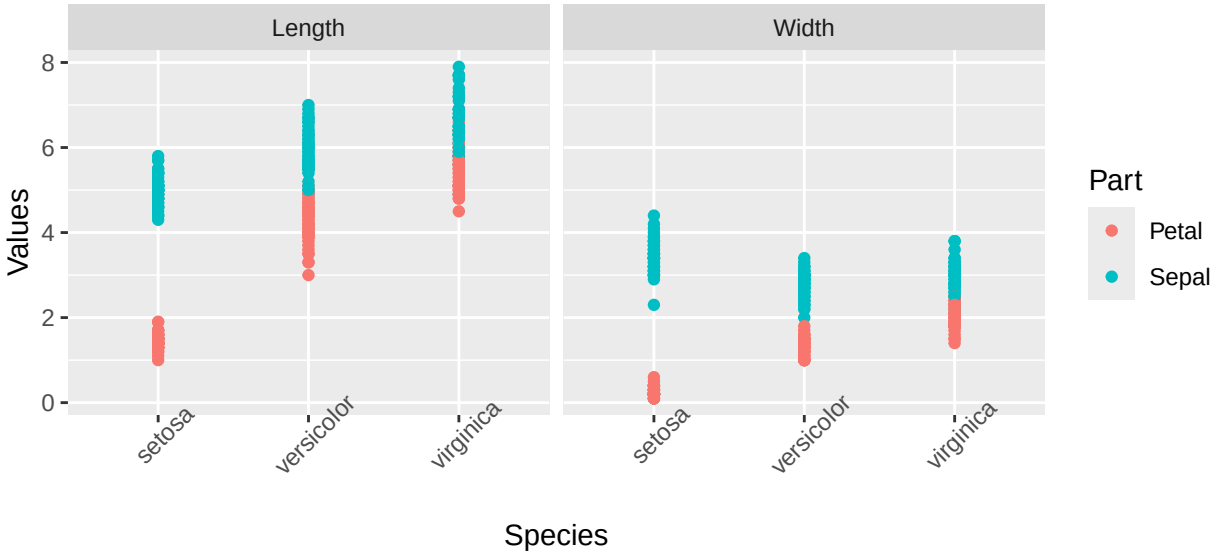
Part C: Use `pivot_wider()` to spread out the "var" column into three new columns containing the type of accident. Your data should now contain two rows per airline (one for each time period), you can check if the first few rows match those printed below.

```
## # A tibble: 6 x 6
##   airline      avail_seat_km_per_week years incidents fatal_accidents fatalities
##   <chr>                         <dbl> <chr>     <int>           <int>      <int>
## 1 Aer Lingus                320906734 1985~         2               0          0
## 2 Aer Lingus                320906734 2000~         0               0          0
## 3 Aeroflot*                1197672318 1985~        76              14        128
## 4 Aeroflot*                1197672318 2000~         6               1         88
## 5 Aerolineas ~              385803648 1985~         6               0          0
## 6 Aerolineas ~              385803648 2000~         1               0          0
```

Question #5: The `iris` data (from the `datasets` package) is a collection of measurements (in cm) of the the sepal and petal dimensions of 50 different flowers coming from 3 different species of iris. These data are frequently attributed to the famous statistician Ronald Fisher

```r
# install.packages("datasets")
data(iris)
```

Your goal in this question is to recreate the following graphic, which requires the use of `tidyr` functions covered in this lab.



- *Note*: you can use the `theme()` function with the argument `axis.text.x = element_text(angle = 45)` to rotate the x-axis labels.