Data Science Wrangling

The textbook for the Data Science course series is freely available online.

Learning Objectives

- How to import data into R from different file formats
- How to scrape data from the web
- How to tidy data using the tidyverse to better facilitate analysis
- How to process strings with regular expressions (regex)
- How to wrangle data using dplyr
- How to work with dates and times as file formats
- How to mine text

Course Overview

Section 1: Data Import

You will learn how to import data from different sources.

Section 2: Tidy Data

You will learn the first pieces of converting data into a tidy format.

Section 3: String Processing

You will learn how to process strings using regular expressions (regex).

Section 4: Dates, Times, and Text Mining

You will learn how to work with dates and times as file formats and how to mine text.

Introduction to Wrangling

The textbook for this section is available here.

Key points

- The first step in data analysis is importing, tidying and cleaning the data. This is the process of data wrangling.
- In this course, we cover several common steps of the data wrangling process: tidying data, string processing, html parsing, working with dates and times, and text mining.

Section 1 Overview

In the **Data Import** section, you will learn how import data into R.

After completing this section, you will be able to:

- **Import** data from spreadsheets.
- Identify and set your **working directory** and specify the **path** to a file.
- Use the **readr** and **readxl** packages to import spreadsheets.
- Use **R-base functions** to import spreadsheets.
- **Download** files from the internet using R.

The textbook for this section is available here.

Importing Spreadsheets

The textbook for this section is available here.

Key points

- Many datasets are stored in spreadsheets. A spreadsheet is essentially a file version of a data frame with rows and columns.
- Spreadsheets have rows separated by returns and columns separated by a delimiter. The most common delimiters are comma, semicolon, white space and tab.
- Many spreadsheets are raw text files and can be read with any basic text editor. However, some formats are proprietary and cannot be read with a text editor, such as Microsoft Excel files (`.xls`).
- Most import functions assume that the first row of a spreadsheet file is a header with column names. To know if the file has a header, it helps to look at the file with a text editor before trying to import it.

Paths and the Working Directory

The textbook for this section is available here.

Key points

- The working directory is where R looks for files and saves files by default.
- See your working directory with `getwd()`. Change your working directory with `setwd()`.
- We suggest you create a directory for each project and keep your raw data inside that directory.
- Use the `file.path()` function to generate a full path from a relative path and a file name. Use `file.path()` instead of `paste()` because `file.path()` is aware of your operating system and will use the correct slashes to navigate your machine.
- The `file.copy()` function copies a file to a new path.

Code

```
# see working directory
getwd()

# change your working directory
setwd()
```

```r
# set path to the location for raw data files in the dslabs package and list files
path <- system.file("extdata", package="dslabs")
list.files(path)
```

```
## [1] "2010_bigfive_regents.xls"
## [2] "carbon_emissions.csv"
## [3] "fertility-two-countries-example.csv"
## [4] "HRlist2.txt"
## [5] "life-expectancy-and-fertility-two-countries-example.csv"
## [6] "murders.csv"
## [7] "olive.csv"
## [8] "RD-Mortality-Report_2015-18-180531.pdf"
## [9] "ssa-death-probability.csv"
```

```r
# generate a full path to a file
filename <- "murders.csv"
fullpath <- file.path(path, filename)
fullpath
```

```
## [1] "/Library/Frameworks/R.framework/Versions/4.0/Resources/library/dslabs/extdata/murders.csv"
```

```r
# copy file from dslabs package to your working directory
file.copy(fullpath, getwd())
```

```
## [1] FALSE
```

```r
# check if the file exists
file.exists(filename)
```

```
## [1] TRUE
```

The readr and readxl Packages

The textbook for this section is available here.

Key points

- **readr** is the **tidyverse** library that includes functions for reading data stored in text file spreadsheets into R. Functions in the package include `read_csv()`, `read_tsv()`, `read_delim()` and more. These differ by the delimiter they use to split columns.
- The **readxl** package provides functions to read Microsoft Excel formatted files.
- The `excel_sheets()` function gives the names of the sheets in the Excel file. These names are passed to the sheet argument for the **readxl** functions `read_excel()`, `read_xls()` and `read_xlsx()`.
- The `read_lines()` function shows the first few lines of a file in R.

Code

```r
if(!require(dslabs)) install.packages("dslabs")
```

```
## Loading required package: dslabs
```

```r
if(!require(tidyverse)) install.packages("tidyverse")
```

```
## Loading required package: tidyverse

## -- Attaching packages -----------------------------------------------------------------------------

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts ----------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
if(!require(readxl)) install.packages("readxl")
```

```
## Loading required package: readxl
```

```r
library(dslabs)
library(tidyverse)    # includes readr
library(readxl)

# inspect the first 3 lines
read_lines("murders.csv", n_max = 3)
```

```
## [1] "state,abb,region,population,total" "Alabama,AL,South,4779736,135"
## [3] "Alaska,AK,West,710231,19"
```

```r
# read file in CSV format
dat <- read_csv(filename)
```

```
## Parsed with column specification:
## cols(
##   state = col_character(),
##   abb = col_character(),
##   region = col_character(),
##   population = col_double(),
##   total = col_double()
## )
```

```r
#read using full path
dat <- read_csv(fullpath)
```

```
## Parsed with column specification:
## cols(
##   state = col_character(),
##   abb = col_character(),
##   region = col_character(),
##   population = col_double(),
##   total = col_double()
## )
```

```r
head(dat)
```

```
## # A tibble: 6 x 5
##   state      abb   region population total
##   <chr>      <chr> <chr>       <dbl> <dbl>
## 1 Alabama    AL    South     4779736   135
## 2 Alaska     AK    West       710231    19
## 3 Arizona    AZ    West      6392017   232
## 4 Arkansas   AR    South     2915918    93
## 5 California CA    West     37253956  1257
## 6 Colorado   CO    West      5029196    65
```

```r
#Ex
path <- system.file("extdata", package = "dslabs")
files <- list.files(path)
files
```

```
## [1] "2010_bigfive_regents.xls"
## [2] "carbon_emissions.csv"
## [3] "fertility-two-countries-example.csv"
## [4] "HRlist2.txt"
## [5] "life-expectancy-and-fertility-two-countries-example.csv"
## [6] "murders.csv"
## [7] "olive.csv"
## [8] "RD-Mortality-Report_2015-18-180531.pdf"
## [9] "ssa-death-probability.csv"
```

```r
filename <- "murders.csv"
filename1 <- "life-expectancy-and-fertility-two-countries-example.csv"
filename2 <- "fertility-two-countries-example.csv"
dat=read.csv(file.path(path, filename))
dat1=read.csv(file.path(path, filename1))
dat2=read.csv(file.path(path, filename2))
```

Importing Data Using R-base Functions

The textbook for this section is available here.

Key point

- R-base import functions (`read.csv()`, `read.table()`, `read.delim()`) generate data frames rather than tibbles and character variables are converted to factors. This can be avoided by setting the argument `stringsAsFactors=FALSE`.

Code

```r
# read.csv converts strings to factors
dat2 <- read.csv(filename)
class(dat2$abb)
```

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

```r
class(dat2$region)
```

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

Downloading Files from the Internet

The textbook for this section is available here.

Key points

- The `read_csv()` function and other import functions can read a URL directly.
- If you want to have a local copy of the file, you can use `download.file()`.
- `tempdir()` creates a directory with a name that is very unlikely not to be unique.
- `tempfile()` creates a character string that is likely to be a unique filename.

Code

```r
url <- "https://raw.githubusercontent.com/rafalab/dslabs/master/inst/extdata/murders.csv"
dat <- read_csv(url)
```

```
## Parsed with column specification:
## cols(
##   state = col_character(),
##   abb = col_character(),
##   region = col_character(),
##   population = col_double(),
##   total = col_double()
## )
```

```r
download.file(url, "murders.csv")
tempfile()
```

```
## [1] "/var/folders/6m/nz2p76pn679b692c99t644bm0000gn/T//Rtmphza1Gy/file36a6178e8391"
```

```r
tmp_filename <- tempfile()
download.file(url, tmp_filename)
dat <- read_csv(tmp_filename)
```

```
## Parsed with column specification:
## cols(
##   state = col_character(),
##   abb = col_character(),
##   region = col_character(),
##   population = col_double(),
##   total = col_double()
## )
```

```r
file.remove(tmp_filename)
```

```
## [1] TRUE
```

Assessment Part 1 - Data Import

1. Which of the following is NOT part of the data wrangling process?

- ☐ A. Importing data into R
- ☐ B. Formatting dates/times
- ☒ C. Checking correlations between your variables
- ☐ D. Tidying data

2. Which files could be opened in a basic text editor?

Select ALL that apply.

- ☒ A. data.txt
- ☒ B. data.csv
- ☐ C. data.xlsx
- ☒ D. data.tsv

3. You want to analyze a file containing race finish times for a recent marathon. You open the file in a basic text editor and see lines that look like the following:

```
initials,state,age,time
vib,MA,61,6:01
adc,TX,45,5:45
kme,CT,50,4:19
```

What type of file is this?

- ☐ A. A comma-delimited file without a header
- ☐ B. A tab-delimited file with a header
- ☐ C. A white space-delimited file without a header
- ☒ D. A comma-delimited file with a header

4. Assume the following is the full path to the directory that a student wants to use as their working directory in R: "/Users/student/Documents/projects/"

Which of the following lines of code CANNOT set the working directory to the desired "projects" directory?

- ☐ A. `setwd("~/Documents/projects/")`
- ☐ B. `setwd("/Users/student/Documents/projects/")`
- ☒ C. `setwd(/Users/student/Documents/projects/)`
- ☐ D. `dir <- "/Users/student/Documents/projects" setwd(dir)`

5. We want to copy the "murders.csv" file from the dslabs package into an existing folder "data", which is located in our HarvardX-Wrangling projects folder. We first enter the code below into our RStudio console.

```
> getwd()
[1] "C:/Users/UNIVERSITY/Documents/Analyses/HarvardX-Wrangling"
> filename <- "murders.csv"
> path <- system.file("extdata", package = "dslabs")
```

Which of the following commands would NOT successfully copy "murders.csv" into the folder "data"?

☒ A.

```
file.copy(file.path(path, "murders.csv"), getwd())
```

☐ B.

```
file.copy(file.path(path, filename), getwd())
```

☐ C.

```
file.copy(file.path(path, "murders.csv"), file.path(getwd(), "data"))
```

☐ D.

```
file.location <- file.path(system.file("extdata", package = "dslabs"), "murders.csv")
file.destination <- file.path(getwd(),"data")
file.copy(file.location, file.destination)
```

6. You are not sure whether the murders.csv file has a header row. How could you check this?

Select ALL that apply.

- ☒ A. Open the file in a basic text editor.
- ☒ B. In the RStudio "Files" pane, click on your file, then select "View File".
- ☒ C. Use the command `read_lines` (remembering to specify the number of rows with the `n_max` argument).

7. What is one difference between `read_excel` and `read_xlsx`?

- ☐ A. `read_excel()` also reads meta-data from the excel file, such as sheet names, while `read_xlsx()` only reads the first sheet in a file.
- ☒ B. `read_excel()` reads both .xls and .xlsx files by detecting the file format from its extension, while `read_xlsx()` only reads .xlsx files.
- ☐ C. `read_excel()` is part of the **readr** package, while `read_xlsx()` is part of the **readxl** package and has more options.
- ☐ D. `read_xlsx()` has been replaced by `read_excel()` in a recent **readxl** package update.

8. You have a file called "times.txt" that contains race finish times for a marathon. The first four lines of the file look like this:

```
initials,state,age,time
vib,MA,61,6:01
adc,TX,45,5:45
kme,CT,50,4:19
```

Which line of code will NOT produce a tibble with column names "initials", "state", "age", and "time"?

- ☐ A. `race_times <- read_csv("times.txt")`
- ☒ B. `race_times <- read.csv("times.txt")`
- ☐ C. `race_times <- read_csv("times.txt", col_names = TRUE)`

☐ D. `race_times <- read_delim("times.txt", delim = ",")`

9. You also have access to marathon finish times in the form of an Excel document named "times.xlsx". In the Excel document, different sheets contain race information for different years. The first sheet is named "2015", the second is named "2016", and the third is named "2017".

Which line of code will NOT import the data contained in the "2016" tab of this Excel sheet?

☐ A. `times_2016 <- read_excel("times.xlsx", sheet = 2)`
☒ B. `times_2016 <- read_xlsx("times.xlsx", sheet = "2")`
☐ C. `times_2016 <- read_excel("times.xlsx", sheet = "2016")`
☐ D. `times_2016 <- read_xlsx("times.xlsx", sheet = 2)`

10. You have a comma-separated values file that contains the initials, home states, ages, and race finish times for marathon runners. The runners' initials contain three characters for the runners' first, middle, and last names (for example, "KME").

You read in the file using the following code.

```
race_times <- read.csv("times.csv")
```

What is the data type of the initials in the object `race_times`?

☐ A. integers
☐ B. characters
☒ C. factors
☐ D. logical

11. Which of the following is NOT a real difference between the readr import functions and the base R import functions?

☐ A. The import functions in the readr package all start as `read_`, while the import functions for base R all start with `read`.
☐ B. Base R import functions automatically convert character columns to factors.
☒ C. The base R import functions can read .csv files, but cannot files with other delimiters, such as .tsv files, or fixed-width files.
☐ D. Base R functions import data as a data frame, while readr functions import data as a tibble.

12. You read in a file containing runner information and marathon finish times using the following code.

```
race_times <- read.csv("times.csv", stringsAsFactors = F)
```

What is the class of the object `race_times`?

☒ A. data frame
☐ B. tibble
☐ C. matrix
☐ D. vector

13. Select the answer choice that summarizes all of the actions that the following lines of code can perform. Please note that the url below is an example and does not lead to data.

```
url <- "https://raw.githubusercontent.com/MyUserName/MyProject/master/MyData.csv "
dat <- read_csv(url)
download.file(url, "MyData.csv")
```

- ☐ A. Create a tibble in R called `dat` that contains the information contained in the csv file stored on Github and save that tibble to the working directory.
- ☐ B. Create a matrix in R called `dat` that contains the information contained in the csv file stored on Github. Download the csv file to the working directory and name the downloaded file "MyData.csv".
- ☐ C. Create a tibble in R called `dat` that contains the information contained in the csv file stored on Github. Download the csv file to the working directory and randomly assign it a temporary name that is very likely to be unique.
- ☒ D. Create a tibble in R called `dat` that contains the information contained in the csv file stored on Github. Download the csv file to the working directory and name the downloaded file "MyData.csv".

Assessment Part 2 - Data Import

14. Inspect the file at the following URL:

https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/datasets/wdbc/wdbc.data

Which **readr** function should be used to import this file?

- ☐ A. `read_table()`
- ☒ B. `read_csv()`
- ☐ C. `read_csv2()`
- ☐ D. `read_tsv()`
- ☐ E. None of the above

15. Check the documentation for the readr function you chose in the previous question to learn about its arguments. Determine which arguments you need to the file from the previous question:

```
url <- "https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/datasets/wdbc/w
```

Does this file have a header row? Does the **readr** function you chose need any additional arguments to import the data correctly?

- ☐ A. Yes, there is a header. No arguments are needed.
- ☐ B. Yes, there is a header. The `header=TRUE` argument is necessary.
- ☐ C. Yes, there is a header. The `col_names=TRUE` argument is necessary.
- ☐ D. No, there is no header. No arguments are needed.
- ☐ E. No, there is no header. The `header=FALSE` argument is necessary.
- ☒ F. No, there is no header. The `col_names=FALSE` argument is necessary.

16. Inspect the imported data from the previous question.

How many rows are in the dataset?

```r
url <- "https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/datasets/wdbc/w
         df <- read_csv(url, col_names = FALSE)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   X2 = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```r
         nrow(df)
```

```
## [1] 569
```

How many columns are in the dataset?

```r
ncol(df)
```

```
## [1] 32
```

Section 2 Overview

In the **Tidy Data** section, you will learn how to convert data from a raw to a tidy format.

This section is divided into three parts: **Reshaping Data**, **Combining Tables**, and **Web Scraping**.

After completing the **Tidy Data** section, you will be able to:

- **Reshape data** using functions from the **tidyr** package, including `gather()`, `spread()`, `separate()`, and `unite()`.
- Combine information from different tables using **join** functions from the **dplyr** package.
- Combine information from different tables using **binding** functions from the **dplyr** package.
- Use **set operators** to combine data frames.
- Gather data from a website through **web scraping** and use of **CSS selectors**.

Tidy Data

The textbook for this section is available here.

Key points

- In tidy data, each row represents an observation and each column represents a different variable.
- In wide data, each row includes several observations and one of the variables is stored in the header.

Code

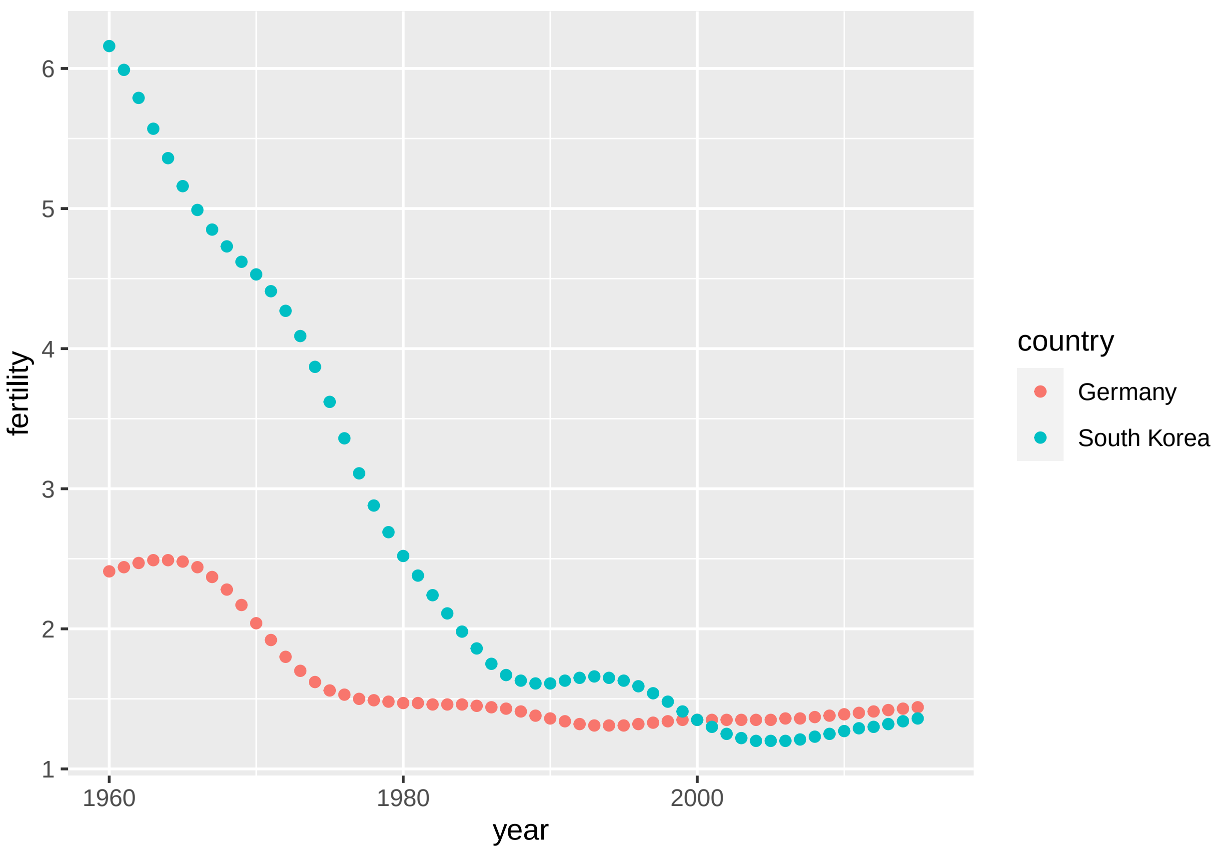
```r
data(gapminder)

# create and inspect a tidy data frame
tidy_data <- gapminder %>%
  filter(country %in% c("South Korea", "Germany")) %>%
  select(country, year, fertility)
head(tidy_data)
```

```
##       country year fertility
## 1     Germany 1960      2.41
## 2 South Korea 1960      6.16
## 3     Germany 1961      2.44
## 4 South Korea 1961      5.99
## 5     Germany 1962      2.47
## 6 South Korea 1962      5.79
```

```r
# plotting tidy data is simple
tidy_data %>%
  ggplot(aes(year, fertility, color = country)) +
  geom_point()
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



```r
# import and inspect example of original Gapminder data in wide format
path <- system.file("extdata", package="dslabs")
filename <- file.path(path,  "fertility-two-countries-example.csv")
wide_data <- read_csv(filename)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   country = col_character()
## )

## See spec(...) for full column specifications.
```

```r
select(wide_data, country, `1960`:`1967`)
```

```
## # A tibble: 2 x 9
##   country     `1960` `1961` `1962` `1963` `1964` `1965` `1966` `1967`
##   <chr>        <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 Germany       2.41   2.44   2.47   2.49   2.49   2.48   2.44   2.37
## 2 South Korea   6.16   5.99   5.79   5.57   5.36   5.16   4.99   4.85
```

Reshaping Data

The textbook for this section is available here, here and here.

Key points

- The **tidyr** package includes several functions that are useful for tidying data.
- The `gather()` function converts wide data into tidy data.
- The `spread()` function converts tidy data to wide data.

Code

```r
# original wide data
path <- system.file("extdata", package="dslabs")
filename <- file.path(path,  "fertility-two-countries-example.csv")
wide_data <- read_csv(filename)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   country = col_character()
## )

## See spec(...) for full column specifications.
```

```r
# tidy data from dslabs
tidy_data <- gapminder %>%
  filter(country %in% c("South Korea", "Germany")) %>%
  select(country, year, fertility)
```

```r
# gather wide data to make new tidy data
new_tidy_data <- wide_data %>%
  gather(year, fertility, `1960`:`2015`)
head(new_tidy_data)
```

```
## # A tibble: 6 x 3
##   country     year  fertility
##   <chr>       <chr>     <dbl>
## 1 Germany     1960       2.41
## 2 South Korea 1960       6.16
## 3 Germany     1961       2.44
## 4 South Korea 1961       5.99
## 5 Germany     1962       2.47
## 6 South Korea 1962       5.79
```

```r
# gather all columns except country
new_tidy_data <- wide_data %>%
  gather(year, fertility, -country)

# gather treats column names as characters by default
class(tidy_data$year)
```

```
## [1] "integer"
```

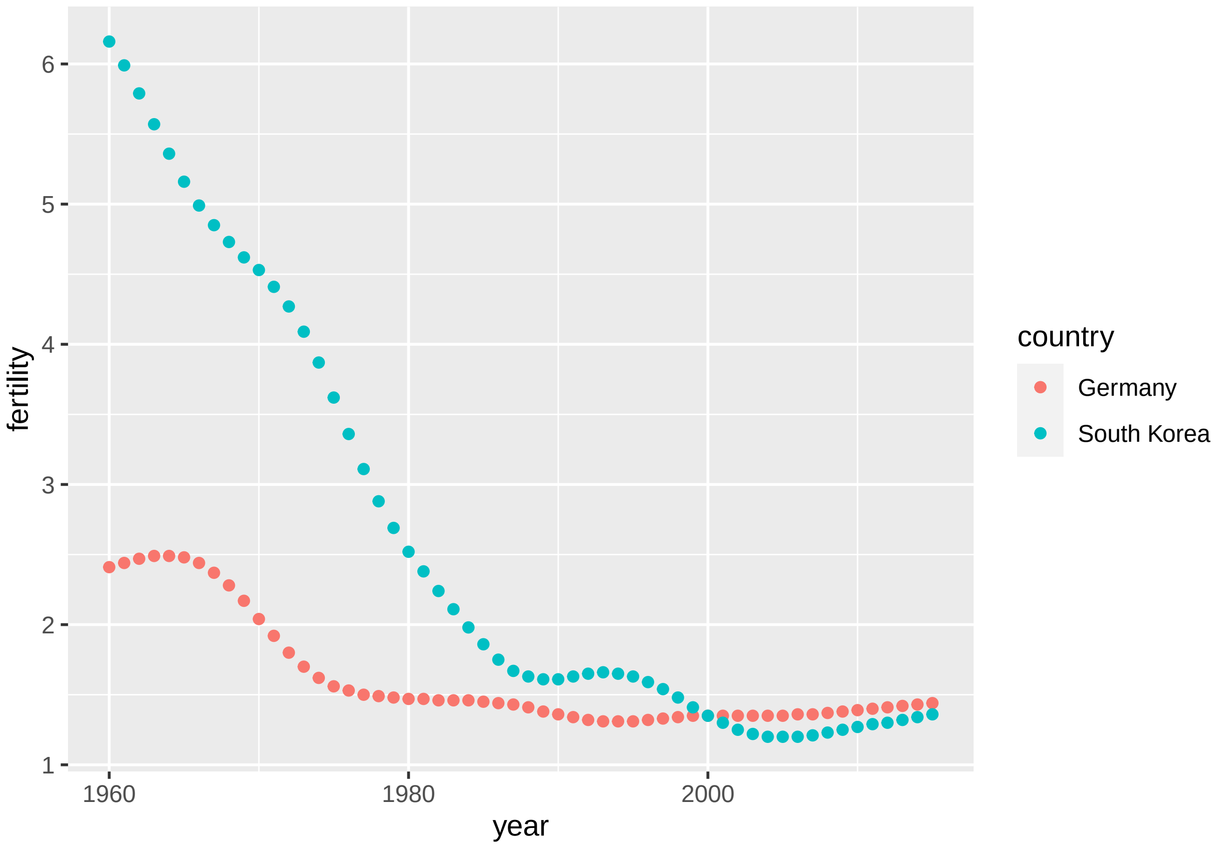
```r
class(new_tidy_data$year)
```

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

```r
# convert gathered column names to numeric
new_tidy_data <- wide_data %>%
  gather(year, fertility, -country, convert = TRUE)
class(new_tidy_data$year)
```

```
## [1] "integer"
```

```r
# ggplot works on new tidy data
new_tidy_data %>%
  ggplot(aes(year, fertility, color = country)) +
  geom_point()
```




```
# spread tidy data to generate wide data
new_wide_data <- new_tidy_data %>% spread(year, fertility)
select(new_wide_data, country, `1960`:`1967`)
```

```
## # A tibble: 2 x 9
##   country     `1960` `1961` `1962` `1963` `1964` `1965` `1966` `1967`
##   <chr>        <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 Germany       2.41   2.44   2.47   2.49   2.49   2.48   2.44   2.37
## 2 South Korea   6.16   5.99   5.79   5.57   5.36   5.16   4.99   4.85
```

Separate and Unite

The textbook for this section is available here and here.

Key points

- The `separate()` function splits one column into two or more columns at a specified character that separates the variables.
- When there is an extra separation in some of the entries, use `fill="right"` to pad missing values with NAs, or use `extra="merge"` to keep extra elements together.
- The `unite()` function combines two columns and adds a separating character.

Code

```r
# import data
path <- system.file("extdata", package = "dslabs")
filename <- file.path(path, "life-expectancy-and-fertility-two-countries-example.csv")
raw_dat <- read_csv(filename)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   country = col_character()
## )

## See spec(...) for full column specifications.
```

```r
select(raw_dat, 1:5)
```

```
## # A tibble: 2 x 5
##   country  `1960_fertility` `1960_life_expec~ `1961_fertility` `1961_life_expec~
##   <chr>               <dbl>             <dbl>            <dbl>             <dbl>
## 1 Germany              2.41              69.3             2.44              69.8
## 2 South K~             6.16              53.0             5.99              53.8
```

```r
# gather all columns except country
dat <- raw_dat %>% gather(key, value, -country)
head(dat)
```

```
## # A tibble: 6 x 3
##   country     key                  value
##   <chr>       <chr>                <dbl>
## 1 Germany     1960_fertility        2.41
## 2 South Korea 1960_fertility        6.16
## 3 Germany     1960_life_expectancy 69.3
## 4 South Korea 1960_life_expectancy 53.0
## 5 Germany     1961_fertility        2.44
## 6 South Korea 1961_fertility        5.99
```

```r
dat$key[1:5]
```

```
## [1] "1960_fertility"       "1960_fertility"       "1960_life_expectancy"
## [4] "1960_life_expectancy" "1961_fertility"
```

```r
# separate on underscores
dat %>% separate(key, c("year", "variable_name"), "_")
```

```
## Warning: Expected 2 pieces. Additional pieces discarded in 112 rows [3, 4, 7, 8,
## 11, 12, 15, 16, 19, 20, 23, 24, 27, 28, 31, 32, 35, 36, 39, 40, ...].

## # A tibble: 224 x 4
##    country     year  variable_name value
##    <chr>       <chr> <chr>         <dbl>
##  1 Germany     1960  fertility      2.41
```

```
##  2 South Korea 1960  fertility      6.16
##  3 Germany     1960  life          69.3
##  4 South Korea 1960  life          53.0
##  5 Germany     1961  fertility      2.44
##  6 South Korea 1961  fertility      5.99
##  7 Germany     1961  life          69.8
##  8 South Korea 1961  life          53.8
##  9 Germany     1962  fertility      2.47
## 10 South Korea 1962  fertility      5.79
## # ... with 214 more rows
```

```
dat %>% separate(key, c("year", "variable_name"))
```

```
## Warning: Expected 2 pieces. Additional pieces discarded in 112 rows [3, 4, 7, 8,
## 11, 12, 15, 16, 19, 20, 23, 24, 27, 28, 31, 32, 35, 36, 39, 40, ...].
```

```
## # A tibble: 224 x 4
##    country     year  variable_name value
##    <chr>       <chr> <chr>         <dbl>
##  1 Germany     1960  fertility      2.41
##  2 South Korea 1960  fertility      6.16
##  3 Germany     1960  life          69.3
##  4 South Korea 1960  life          53.0
##  5 Germany     1961  fertility      2.44
##  6 South Korea 1961  fertility      5.99
##  7 Germany     1961  life          69.8
##  8 South Korea 1961  life          53.8
##  9 Germany     1962  fertility      2.47
## 10 South Korea 1962  fertility      5.79
## # ... with 214 more rows
```

```
# split on all underscores, pad empty cells with NA
dat %>% separate(key, c("year", "first_variable_name", "second_variable_name"),
                 fill = "right")
```

```
## # A tibble: 224 x 5
##    country     year  first_variable_name second_variable_name value
##    <chr>       <chr> <chr>               <chr>                <dbl>
##  1 Germany     1960  fertility           <NA>                  2.41
##  2 South Korea 1960  fertility           <NA>                  6.16
##  3 Germany     1960  life                expectancy           69.3
##  4 South Korea 1960  life                expectancy           53.0
##  5 Germany     1961  fertility           <NA>                  2.44
##  6 South Korea 1961  fertility           <NA>                  5.99
##  7 Germany     1961  life                expectancy           69.8
##  8 South Korea 1961  life                expectancy           53.8
##  9 Germany     1962  fertility           <NA>                  2.47
## 10 South Korea 1962  fertility           <NA>                  5.79
## # ... with 214 more rows
```

```r
# split on first underscore but keep life_expectancy merged
dat %>% separate(key, c("year", "variable_name"), sep = "_", extra = "merge")
```

```
## # A tibble: 224 x 4
##    country     year  variable_name   value
##    <chr>       <chr> <chr>           <dbl>
##  1 Germany     1960  fertility        2.41
##  2 South Korea 1960  fertility        6.16
##  3 Germany     1960  life_expectancy 69.3
##  4 South Korea 1960  life_expectancy 53.0
##  5 Germany     1961  fertility        2.44
##  6 South Korea 1961  fertility        5.99
##  7 Germany     1961  life_expectancy 69.8
##  8 South Korea 1961  life_expectancy 53.8
##  9 Germany     1962  fertility        2.47
## 10 South Korea 1962  fertility        5.79
## # ... with 214 more rows
```

```r
# separate then spread
dat %>% separate(key, c("year", "variable_name"), sep = "_", extra = "merge") %>%
  spread(variable_name, value)
```

```
## # A tibble: 112 x 4
##    country year  fertility life_expectancy
##    <chr>   <chr>     <dbl>           <dbl>
##  1 Germany 1960       2.41            69.3
##  2 Germany 1961       2.44            69.8
##  3 Germany 1962       2.47            70.0
##  4 Germany 1963       2.49            70.1
##  5 Germany 1964       2.49            70.7
##  6 Germany 1965       2.48            70.6
##  7 Germany 1966       2.44            70.8
##  8 Germany 1967       2.37            71.0
##  9 Germany 1968       2.28            70.6
## 10 Germany 1969       2.17            70.5
## # ... with 102 more rows
```

```r
# separate then unite
dat %>%
  separate(key, c("year", "first_variable_name", "second_variable_name"), fill = "right") %>%
  unite(variable_name, first_variable_name, second_variable_name, sep="_")
```

```
## # A tibble: 224 x 4
##    country     year  variable_name   value
##    <chr>       <chr> <chr>           <dbl>
##  1 Germany     1960  fertility_NA     2.41
##  2 South Korea 1960  fertility_NA     6.16
##  3 Germany     1960  life_expectancy 69.3
##  4 South Korea 1960  life_expectancy 53.0
##  5 Germany     1961  fertility_NA     2.44
##  6 South Korea 1961  fertility_NA     5.99
##  7 Germany     1961  life_expectancy 69.8
```

```
## 8 South Korea 1961  life_expectancy 53.8
## 9 Germany     1962  fertility_NA     2.47
## 10 South Korea 1962  fertility_NA     5.79
## # ... with 214 more rows
```

```r
# full code for tidying data
dat %>%
  separate(key, c("year", "first_variable_name", "second_variable_name"), fill = "right") %>%
  unite(variable_name, first_variable_name, second_variable_name, sep="_") %>%
  spread(variable_name, value) %>%
  rename(fertility = fertility_NA)
```

```
## # A tibble: 112 x 4
##    country year  fertility life_expectancy
##    <chr>   <chr>     <dbl>           <dbl>
##  1 Germany 1960       2.41            69.3
##  2 Germany 1961       2.44            69.8
##  3 Germany 1962       2.47            70.0
##  4 Germany 1963       2.49            70.1
##  5 Germany 1964       2.49            70.7
##  6 Germany 1965       2.48            70.6
##  7 Germany 1966       2.44            70.8
##  8 Germany 1967       2.37            71.0
##  9 Germany 1968       2.28            70.6
## 10 Germany 1969       2.17            70.5
## # ... with 102 more rows
```

Assessment Part 1 - Reshaping Data

1. A collaborator sends you a file containing data for three years of average race finish times.

```
age_group,2015,2016,2017
20,3:46,3:22,3:50
30,3:50,3:43,4:43
40,4:39,3:49,4:51
50,4:48,4:59,5:01
```

Are these data considered "tidy" in R? Why or why not?

- ☐ A. Yes. These data are considered "tidy" because each row contains unique observations.
- ☐ B. Yes. These data are considered "tidy" because there are no missing data in the data frame.
- ☒ C. No. These data are not considered "tidy" because the variable "year" is stored in the header.
- ☐ D. No. These data are not considered "tidy" because there are not an equal number of columns and rows.

2. Below are four versions of the same dataset. Which one is in a tidy format?

☒ A.

```
state      abb  region  population total
Alabama    AL   South   4779736    135
Alaska     AK   West    710231     19
Arizona    AZ   West    6392017    232
Arkansas   AR   South   2915918    93
California CA   West    37253956   1257
Colorado   CO   West    5029196    65
```

☐ B.

```
state    abb  region        var        people
Alabama  AL   South         population 4779736
Alabama  AL   South         total      135
Alaska   AK   West          population 710231
Alaska   AK   West          total      19
Arizona  AZ   West          population 6392017
Arizona  AZ   West          total      232
```

☐ C.

```
state       abb       Northeast    South      North Central    West
Alabama     AL        NA           4779736    NA               NA
Alaska      AK        NA           NA         NA               710231
Arizona     AZ        NA           NA         NA               6392017
Arkansas    AR        NA           2915918    NA               NA
California  CA        NA           NA         NA               37253956
Colorado    CO        NA           NA         NA               5029196
```

☐ D.

```
state       abb  region    rate
Alabama     AL   South     2.82e-05
Alaska      AK   West      2.68e-05
Arizona     AZ   West      3.63e-05
Arkansas    AR   South     3.19e-05
California  CA   West      3.37e-05
Colorado    CO   West      1.29e-05
```

3. Your file called “times.csv” has age groups and average race finish times for three years of marathons.

```
age_group,2015,2016,2017
20,3:46,3:22,3:50
30,3:50,3:43,4:43
40,4:39,3:49,4:51
50,4:48,4:59,5:01
```

You read in the data file using the following command.

```
d <- read_csv("files/times.csv")
```

```
## Parsed with column specification:
## cols(
##   age_group = col_double(),
##   `2015` = col_time(format = ""),
##   `2016` = col_time(format = ""),
##   `2017` = col_time(format = "")
## )
```

Which commands will help you "tidy" the data?

```r
tidy_data <- d %>%
gather(year, time, `2015`:`2017`)
tidy_data
```

```
## # A tibble: 12 x 3
##    age_group year  time
##        <dbl> <chr> <time>
##  1        20 2015  03:46
##  2        30 2015  03:50
##  3        40 2015  04:39
##  4        50 2015  04:48
##  5        20 2016  03:22
##  6        30 2016  03:43
##  7        40 2016  03:49
##  8        50 2016  04:59
##  9        20 2017  03:50
## 10        30 2017  04:43
## 11        40 2017  04:51
## 12        50 2017  05:01
```

☒ A.

```r
tidy_data <- d %>%
gather(year, time, `2015`:`2017`)
```

☐ B.

```r
tidy_data <- d %>%
spread(year, time, `2015`:`2017`)
```

☐ C.

```r
tidy_data <- d %>%
gather(age_group, year, time, `2015`:`2017`)
```

☐ D.

```r
tidy_data <- d %>%
gather(time, `2015`:`2017`)
```

4. You have a dataset on U.S. contagious diseases, but it is in the following wide format:

```
> head(dat_wide)
state   year    population      Hepatitis A Mumps Polio Rubella
Alabama 1990    4040587         86          19    76    1
Alabama 1991    4066003         39          14    65    0
Alabama 1992    4097169         35          12    24    0
Alabama 1993    4133242         40          22    67    0
Alabama 1994    4173361         72          12    39    0
Alabama 1995    4216645         75          2     38    0
```

Which of the following would transform this into a tidy dataset, with each row representing an observation of the incidence of each specific disease (as shown below)?

```
> head(dat_tidy)
state   year    population   disease      count
Alabama 1990    4040587      Hepatitis A  86
Alabama 1991    4066003      Hepatitis A  39
Alabama 1992    4097169      Hepatitis A  35
Alabama 1993    4133242      Hepatitis A  40
Alabama 1994    4173361      Hepatitis A  72
Alabama 1995    4216645      Hepatitis A  75
```

- ☐ A.

```
dat_tidy <- dat_wide %>%
gather (key = count, value = disease, `Hepatitis A`, `Rubella`)
```

- ☐ B.

```
dat_tidy <- dat_wide %>%
gather(key = count, value = disease, -state, -year, -population)
```

- ☐ C.

```
dat_tidy <- dat_wide %>%
gather(key = disease, value = count, -state)
```

- ☒ D.

```
dat_tidy <- dat_wide %>%
gather(key = disease, value = count, "Hepatitis A": "Rubella")
```

5. You have successfully formatted marathon finish times into a tidy object called `tidy_data`. The first few lines are shown below.

```
age_group year  time
20        2015  03:46
30        2015  03:50
40        2015  04:39
50        2015  04:48
20        2016  03:22
```

Select the code that converts these data back to the wide format, where each year has a separate column.

```
tidy_data %>% spread(year, time)
```

```
## # A tibble: 4 x 4
##   age_group `2015` `2016` `2017`
##       <dbl> <time> <time> <time>
## 1        20 03:46  03:22  03:50
## 2        30 03:50  03:43  04:43
## 3        40 04:39  03:49  04:51
## 4        50 04:48  04:59  05:01
```

- ☐ A. `tidy_data %>% spread(time, year)`
- ☒ B. `tidy_data %>% spread(year, time)`
- ☐ C. `tidy_data %>% spread(year, age_group)`
- ☐ D. ``tidy_data %>% spread(time, year, `2015`:`2017`)``

6. You have the following dataset:

```
> head(dat)
state    abb  region        var         people
Alabama  AL   South         population  4779736
Alabama  AL   South         total       135
Alaska   AK   West          population  710231
Alaska   AK   West          total       19
Arizona  AZ   West          population  6392017
Arizona  AZ   West          total       232
```

You would like to transform it into a dataset where population and total are each their own column (shown below). Which code would best accomplish this?

```
state       abb  region  population total
Alabama     AL   South   4779736    135
Alaska      AK   West    710231     19
Arizona     AZ   West    6392017    232
Arkansas    AR   South   2915918    93
California  CA   West    37253956   1257
Colorado    CO   West    5029196    65
```

- ☒ A. `dat_tidy <- dat %>% spread(key = var, value = people)`
- ☐ B. `dat_tidy <- dat %>% spread(key = state:region, value = people)`
- ☐ C. `dat_tidy <- dat %>% spread(key = people, value = var)`
- ☐ D. `dat_tidy <- dat %>% spread(key = region, value = people)`

7. A collaborator sends you a file containing data for two years of average race finish times, "times2.csv":.

```
age_group,2015_time,2015_participants,2016_time,2016_participants
20,3:46,54,3:22,62
30,3:50,60,3:43,58
40,4:39,29,3:49,33
50,4:48,10,4:59,14
```

You read in the data file

```
d <- read_csv("files/times2.csv")
```

```
## Parsed with column specification:
## cols(
##   age_group = col_double(),
##   `2015_time` = col_time(format = ""),
##   `2015_participants` = col_double(),
##   `2016_time` = col_time(format = ""),
##   `2016_participants` = col_double()
## )
```

Which of the answers below best tidys the data?

```
tidy_data <- d %>%
        gather(key = "key", value = "value", -age_group) %>%
    separate(col = key, into = c("year", "variable_name"), sep = "_") %>%
    spread(key = variable_name, value = value)
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
tidy_data
```

```
## # A tibble: 8 x 4
##   age_group year  participants  time
##       <dbl> <chr>        <dbl> <dbl>
## 1        20 2015            54 13560
## 2        20 2016            62 12120
## 3        30 2015            60 13800
## 4        30 2016            58 13380
## 5        40 2015            29 16740
## 6        40 2016            33 13740
## 7        50 2015            10 17280
## 8        50 2016            14 17940
```

☐ A.

```
tidy_data <- d %>%
    gather(key = "key", value = "value", -age_group) %>%
    separate(col = key, into = c("year", "variable_name"), sep = ".") %>%
    spread(key = variable_name, value = value)
```

☒ B.

```
tidy_data <- d %>%
        gather(key = "key", value = "value", -age_group) %>%
    separate(col = key, into = c("year", "variable_name"), sep = "_") %>%
    spread(key = variable_name, value = value)
```

☐ C.

```
 tidy_data <- d %>%
        gather(key = "key", value = "value") %>%
    separate(col = key, into = c("year", "variable_name"), sep = "_") %>%
    spread(key = variable_name, value = value)
```

☐ D.

```
tidy_data <- d %>%
        gather(key = "key", value = "value", -age_group) %>%
    separate(col = key, into = "year", sep = "_") %>%
    spread(key = year, value = value)
```

8. You are in the process of tidying some data on heights, hand length, and wingspan for basketball players in the draft. Currently, you have the following:

```
> head(stats)
key                   value
allen_height          75
allen_hand_length     8.25
allen_wingspan        79.25
bamba_height          83.25
bamba_hand_length     9.75
bamba_wingspan        94
```

Select all of the correct commands below that would turn this data into a "tidy" format.

☒ A.

```
tidy_data <- stats %>%
    separate(col = key, into = c("player", "variable_name"), sep = "_", extra = "merge") %>%
    spread(key = variable_name, value = value)
```

☐ B.

```
tidy_data <- stats %>%
    separate(col = key, into = c("player", "variable_name1", "variable_name2"), sep = "_", fill = "righ
    unite(col = variable_name, variable_name1, variable_name2, sep = "_") %>%
    spread(key = variable_name, value = value)
```

☐ C.

```
tidy_data <- stats %>%
    separate(col = key, into = c("player", "variable_name"), sep = "_") %>%
    spread(key = variable_name, value = value)
```

Assessment Part 2 - Reshaping Data

9. Examine the built-in dataset `co2`. This dataset comes with base R, not **dslabs** - just type `co2` to access the dataset.

```
co2
```

```
##         Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct
## 1959 315.42 316.31 316.50 317.56 318.13 318.00 316.39 314.65 313.68 313.18
## 1960 316.27 316.81 317.42 318.87 319.87 319.43 318.01 315.74 314.00 313.68
## 1961 316.73 317.54 318.38 319.31 320.42 319.61 318.42 316.63 314.83 315.16
## 1962 317.78 318.40 319.53 320.42 320.85 320.45 319.45 317.25 316.11 315.27
## 1963 318.58 318.92 319.70 321.22 322.08 321.31 319.58 317.61 316.05 315.83
## 1964 319.41 320.07 320.74 321.40 322.06 321.73 320.27 318.54 316.54 316.71
## 1965 319.27 320.28 320.73 321.97 322.00 321.71 321.05 318.71 317.66 317.14
## 1966 320.46 321.43 322.23 323.54 323.91 323.59 322.24 320.20 318.48 317.94
## 1967 322.17 322.34 322.88 324.25 324.83 323.93 322.38 320.76 319.10 319.24
## 1968 322.40 322.99 323.73 324.86 325.40 325.20 323.98 321.95 320.18 320.09
## 1969 323.83 324.26 325.47 326.50 327.21 326.54 325.72 323.50 322.22 321.62
## 1970 324.89 325.82 326.77 327.97 327.91 327.50 326.18 324.53 322.93 322.90
## 1971 326.01 326.51 327.01 327.62 328.76 328.40 327.20 325.27 323.20 323.40
## 1972 326.60 327.47 327.58 329.56 329.90 328.92 327.88 326.16 324.68 325.04
## 1973 328.37 329.40 330.14 331.33 332.31 331.90 330.70 329.15 327.35 327.02
## 1974 329.18 330.55 331.32 332.48 332.92 332.08 331.01 329.23 327.27 327.21
## 1975 330.23 331.25 331.87 333.14 333.80 333.43 331.73 329.90 328.40 328.17
## 1976 331.58 332.39 333.33 334.41 334.71 334.17 332.89 330.77 329.14 328.78
## 1977 332.75 333.24 334.53 335.90 336.57 336.10 334.76 332.59 331.42 330.98
## 1978 334.80 335.22 336.47 337.59 337.84 337.72 336.37 334.51 332.60 332.38
## 1979 336.05 336.59 337.79 338.71 339.30 339.12 337.56 335.92 333.75 333.70
## 1980 337.84 338.19 339.91 340.60 341.29 341.00 339.39 337.43 335.72 335.84
## 1981 339.06 340.30 341.21 342.33 342.74 342.08 340.32 338.26 336.52 336.68
## 1982 340.57 341.44 342.53 343.39 343.96 343.18 341.88 339.65 337.81 337.69
## 1983 341.20 342.35 342.93 344.77 345.58 345.14 343.81 342.21 339.69 339.82
## 1984 343.52 344.33 345.11 346.88 347.25 346.62 345.22 343.11 340.90 341.18
## 1985 344.79 345.82 347.25 348.17 348.74 348.07 346.38 344.51 342.92 342.62
## 1986 346.11 346.78 347.68 349.37 350.03 349.37 347.76 345.73 344.68 343.99
## 1987 347.84 348.29 349.23 350.80 351.66 351.07 349.33 347.92 346.27 346.18
## 1988 350.25 351.54 352.05 353.41 354.04 353.62 352.22 350.27 348.55 348.72
## 1989 352.60 352.92 353.53 355.26 355.52 354.97 353.75 351.52 349.64 349.83
## 1990 353.50 354.55 355.23 356.04 357.00 356.07 354.67 352.76 350.82 351.04
## 1991 354.59 355.63 357.03 358.48 359.22 358.12 356.06 353.92 352.05 352.11
## 1992 355.88 356.63 357.72 359.07 359.58 359.17 356.94 354.92 352.94 353.23
## 1993 356.63 357.10 358.32 359.41 360.23 359.55 357.53 355.48 353.67 353.95
## 1994 358.34 358.89 359.95 361.25 361.67 360.94 359.55 357.49 355.84 356.00
## 1995 359.98 361.03 361.66 363.48 363.82 363.30 361.94 359.50 358.11 357.80
## 1996 362.09 363.29 364.06 364.76 365.45 365.01 363.70 361.54 359.51 359.65
## 1997 363.23 364.06 364.61 366.40 366.84 365.68 364.52 362.57 360.24 360.83
##         Nov    Dec
## 1959 314.66 315.43
## 1960 314.84 316.03
## 1961 315.94 316.85
## 1962 316.53 317.53
## 1963 316.91 318.20
## 1964 317.53 318.55
```

```
## 1965 318.70 319.25
## 1966 319.63 320.87
## 1967 320.56 321.80
## 1968 321.16 322.74
## 1969 322.69 323.95
## 1970 323.85 324.96
## 1971 324.63 325.85
## 1972 326.34 327.39
## 1973 327.99 328.48
## 1974 328.29 329.41
## 1975 329.32 330.59
## 1976 330.14 331.52
## 1977 332.24 333.68
## 1978 333.75 334.78
## 1979 335.12 336.56
## 1980 336.93 338.04
## 1981 338.19 339.44
## 1982 339.09 340.32
## 1983 340.98 342.82
## 1984 342.80 344.04
## 1985 344.06 345.38
## 1986 345.48 346.72
## 1987 347.64 348.78
## 1988 349.91 351.18
## 1989 351.14 352.37
## 1990 352.69 354.07
## 1991 353.64 354.89
## 1992 354.09 355.33
## 1993 355.30 356.78
## 1994 357.59 359.05
## 1995 359.61 360.74
## 1996 360.80 362.38
## 1997 362.49 364.34
```

Is `co2` tidy? Why or why not?

- ☐ A. `co2` is tidy data: it has one year for each row.
- ☐ B. `co2` is tidy data: each column is a different month.
- ☐ C. `co2` is not tidy: there are multiple observations per column.
- ☒ D. `co2` is not tidy: to be tidy we would have to wrangle it to have three columns (year, month and value), and then each co2 observation would have a row.

10. Run the following command to define the co2_wide object:

```
co2_wide <- data.frame(matrix(co2, ncol = 12, byrow = TRUE)) %>%
      setNames(1:12) %>%
    mutate(year = as.character(1959:1997))
```

Use the `gather()` function to make this dataset tidy. Call the column with the CO2 measurements `co2` and call the month column `month`. Name the resulting object `co2_tidy`.

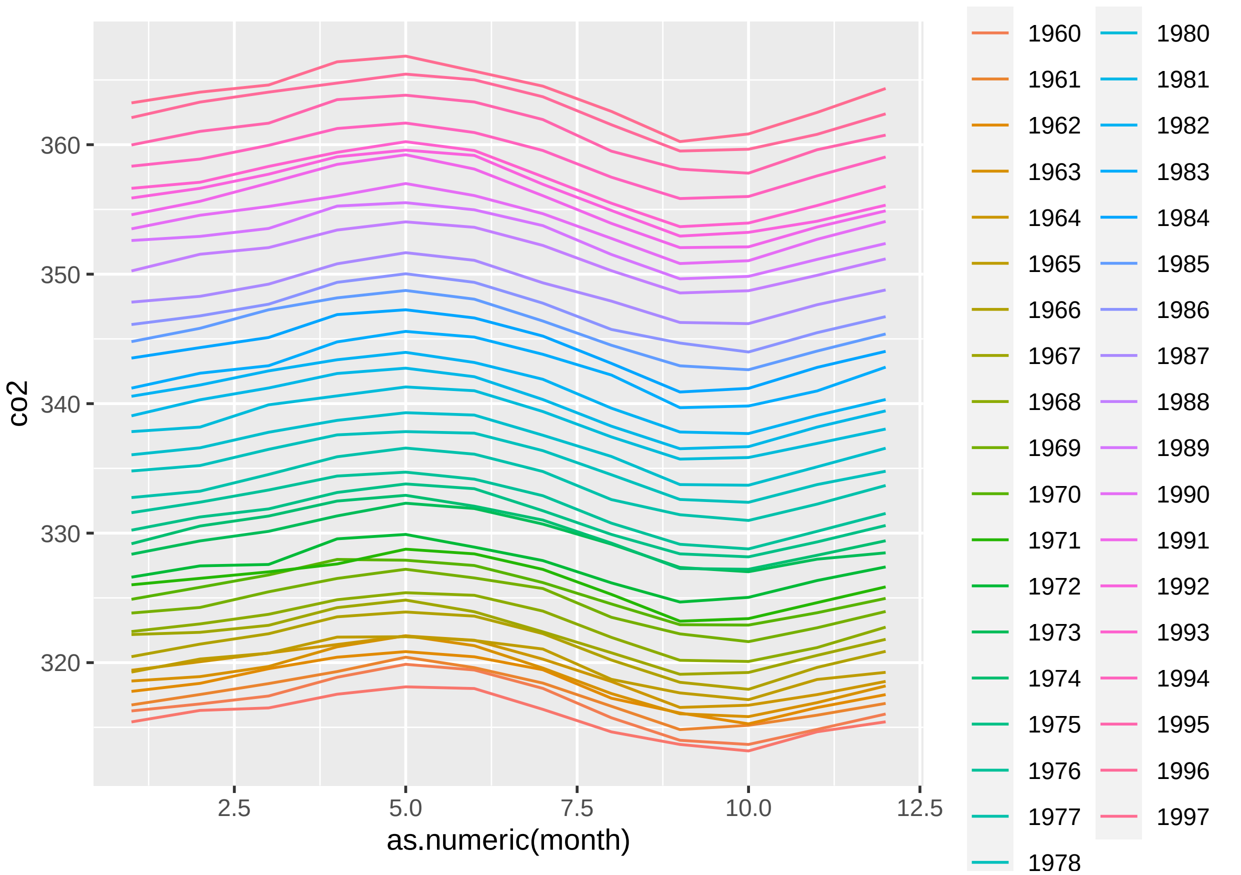
Which code would return the correct tidy format?

```
co2_tidy <- gather(co2_wide,month,co2,-year)
```

- ☐ A. `co2_tidy <- gather(co2_wide,month,co2,year)`
- ☐ B. `co2_tidy <- gather(co2_wide,co2,month,-year)`
- ☐ C. `co2_tidy <- gather(co2_wide,co2,month,year)`
- ☒ D. `co2_tidy <- gather(co2_wide,month,co2,-year)`

11. Use `co2_tidy` to plot CO2 versus month with a different curve for each year:

```
co2_tidy %>% ggplot(aes(as.numeric(month), co2, color = year)) + geom_line()
```



What can be concluded from this plot?

- ☐ A. CO2 concentrations increased monotonically (never decreased) from 1959 to 1997.
- ☒ B. CO2 concentrations are highest around May and the yearly average increased from 1959 to 1997.
- ☐ C. CO2 concentrations are highest around October and the yearly average increased from 1959 to 1997.
- ☐ D. Yearly average CO2 concentrations have remained constant over time.
- ☐ E. CO2 concentrations do not have a seasonal trend.

12. Load the `admissions` dataset from **dslabs**, which contains college admission information for men and women across six majors, and remove the `applicants` percentage column:

```r
data(admissions)
dat <- admissions %>% select(-applicants)
```

Your goal is to get the data in the shape that has one row for each major, like this:

```
major  men   women
A       62    82
B       63    68
C       37    34
D       33    35
E       28    24
F        6     7
```

Which command could help you to wrangle the data into the desired format?

```r
dat_tidy <- spread(dat, gender, admitted)
```

- ☐ A. `dat_tidy <- spread(dat, major, admitted)`
- ☐ B. `dat_tidy <- spread(dat, gender, major)`
- ☒ C. `dat_tidy <- spread(dat, gender, admitted)`
- ☐ D. `dat_tidy <- spread(dat, admitted, gender)`

13. Now use the `admissions` dataset to create the object `tmp`, which has columns `major`, `gender`, `key` and `value`:

```r
tmp <- gather(admissions, key, value, admitted:applicants)
tmp
```

```
##    major gender        key value
## 1      A    men   admitted    62
## 2      B    men   admitted    63
## 3      C    men   admitted    37
## 4      D    men   admitted    33
## 5      E    men   admitted    28
## 6      F    men   admitted     6
## 7      A  women   admitted    82
## 8      B  women   admitted    68
## 9      C  women   admitted    34
## 10     D  women   admitted    35
## 11     E  women   admitted    24
## 12     F  women   admitted     7
## 13     A    men applicants   825
## 14     B    men applicants   560
## 15     C    men applicants   325
## 16     D    men applicants   417
## 17     E    men applicants   191
## 18     F    men applicants   373
## 19     A  women applicants   108
## 20     B  women applicants    25
## 21     C  women applicants   593
## 22     D  women applicants   375
## 23     E  women applicants   393
## 24     F  women applicants   341
```

Combine the key and gender and create a new column called `column_name` to get a variable with the following values: `admitted_men`, `admitted_women`, `applicants_men`and `applicants_women`. Save the new data as `tmp2`.

Which command could help you to wrangle the data into the desired format?

```
tmp2 <- unite(tmp, column_name, c(key, gender))
```

- ☐ A. `tmp2 <- spread(tmp, column_name, key, gender)`
- ☐ B. `tmp2 <- gather(tmp, column_name, c(gender,key))`
- ☐ C. `tmp2 <- unite(tmp, column_name, c(gender, key))`
- ☐ D. `tmp2 <- spread(tmp, column_name, c(key,gender))`
- ☒ E. `tmp2 <- unite(tmp, column_name, c(key, gender))`

14. Which function can reshape `tmp2` to a table with six rows and five columns named `major`, `admitted_men`, `admitted_women`, `applicants_men` and `applicants_women`?

```
spread(tmp2, column_name, value)
```

```
##   major admitted_men admitted_women applicants_men applicants_women
## 1     A           62             82            825              108
## 2     B           63             68            560               25
## 3     C           37             34            325              593
## 4     D           33             35            417              375
## 5     E           28             24            191              393
## 6     F            6              7            373              341
```

- ☐ A. `gather()`
- ☒ B. `spread()`
- ☐ C. `separate()`
- ☐ D. `unite()`

Combining Tables

The textbook for this section is available here.

Key points

- The join functions in the **dplyr** package combine two tables such that matching rows are together.
- `left_join()` only keeps rows that have information in the first table.
- `right_join()` only keeps rows that have information in the second table.
- `inner_join()` only keeps rows that have information in both tables.
- `full_join()` keeps all rows from both tables.
- `semi_join()` keeps the part of first table for which we have information in the second.
- `anti_join()` keeps the elements of the first table for which there is no information in the second.

Code

```
if(!require(ggrepel)) install.packages("ggrepel")
```

```
## Loading required package: ggrepel
```

```r
# import US murders data
library(ggrepel)
ds_theme_set()
data(murders)
head(murders)
```

```
##        state abb region population total
## 1    Alabama  AL  South    4779736   135
## 2     Alaska  AK   West     710231    19
## 3    Arizona  AZ   West    6392017   232
## 4   Arkansas  AR  South    2915918    93
## 5 California  CA   West   37253956  1257
## 6   Colorado  CO   West    5029196    65
```

```r
# import US election results data
data(polls_us_election_2016)
head(results_us_election_2016)
```

```
##          state electoral_votes clinton trump others
## 1   California              55    61.7  31.6    6.7
## 2        Texas              38    43.2  52.2    4.5
## 3      Florida              29    47.8  49.0    3.2
## 4     New York              29    59.0  36.5    4.5
## 5     Illinois              20    55.8  38.8    5.4
## 6 Pennsylvania              20    47.9  48.6    3.6
```

```r
identical(results_us_election_2016$state, murders$state)
```

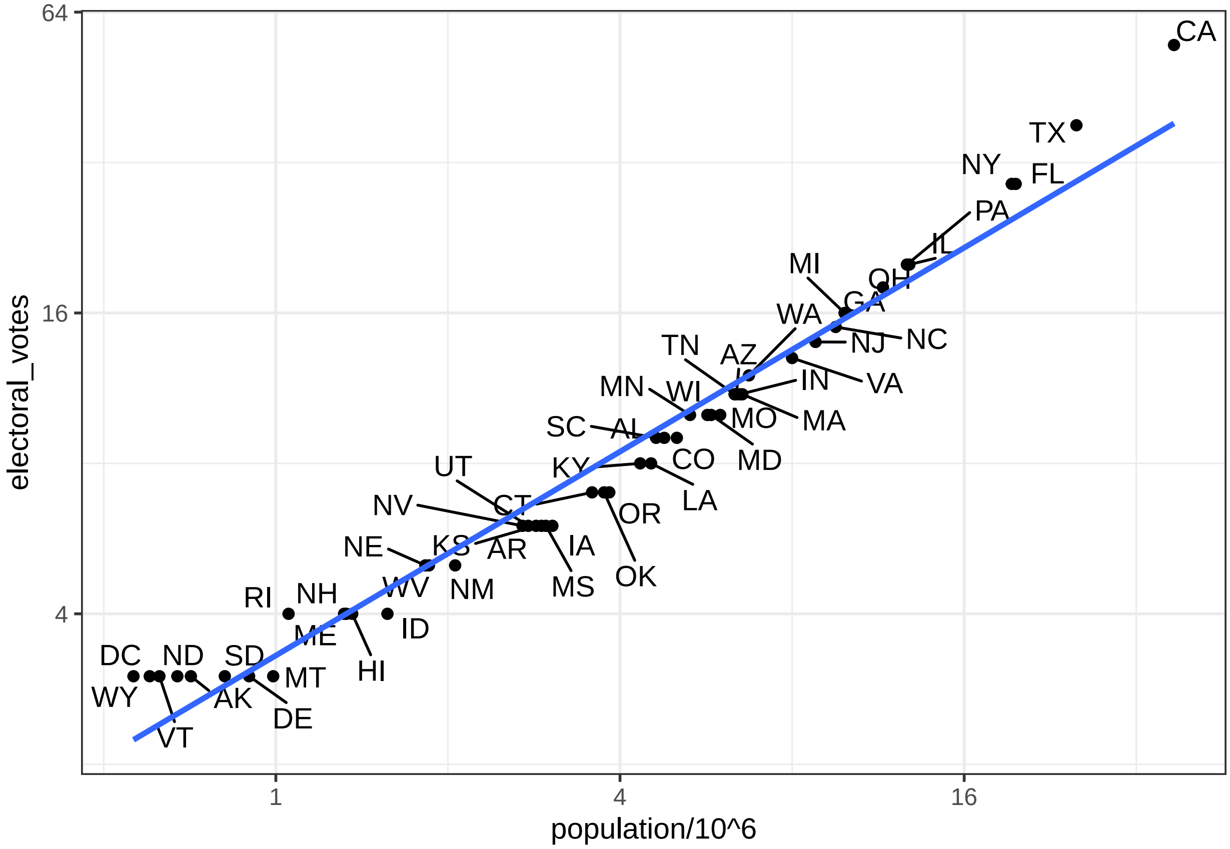
```
## [1] FALSE
```

```r
# join the murders table and US election results table
tab <- left_join(murders, results_us_election_2016, by = "state")
head(tab)
```

```
##        state abb region population total electoral_votes clinton trump others
## 1    Alabama  AL  South    4779736   135               9    34.4  62.1    3.6
## 2     Alaska  AK   West     710231    19               3    36.6  51.3   12.2
## 3    Arizona  AZ   West    6392017   232              11    45.1  48.7    6.2
## 4   Arkansas  AR  South    2915918    93               6    33.7  60.6    5.8
## 5 California  CA   West   37253956  1257              55    61.7  31.6    6.7
## 6   Colorado  CO   West    5029196    65               9    48.2  43.3    8.6
```

```r
# plot electoral votes versus population
tab %>% ggplot(aes(population/10^6, electoral_votes, label = abb)) +
  geom_point() +
  geom_text_repel() +
  scale_x_continuous(trans = "log2") +
  scale_y_continuous(trans = "log2") +
  geom_smooth(method = "lm", se = FALSE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```r
# make two smaller tables to demonstrate joins
tab1 <- slice(murders, 1:6) %>% select(state, population)
tab1
```

```
##        state population
## 1    Alabama    4779736
## 2     Alaska     710231
## 3    Arizona    6392017
## 4   Arkansas    2915918
## 5 California   37253956
## 6   Colorado    5029196
```

```r
tab2 <- slice(results_us_election_2016, c(1:3, 5, 7:8)) %>% select(state, electoral_votes)
tab2
```

```
##        state electoral_votes
## 1 California              55
## 2      Texas              38
## 3    Florida              29
## 4   Illinois              20
## 5       Ohio              18
## 6    Georgia              16
```

```r
# experiment with different joins
left_join(tab1, tab2)
```

```
## Joining, by = "state"
```

```
##        state population electoral_votes
## 1    Alabama    4779736              NA
## 2     Alaska     710231              NA
## 3    Arizona    6392017              NA
## 4   Arkansas    2915918              NA
## 5 California   37253956              55
## 6   Colorado    5029196              NA
```

```r
tab1 %>% left_join(tab2)
```

```
## Joining, by = "state"
```

```
##        state population electoral_votes
## 1    Alabama    4779736              NA
## 2     Alaska     710231              NA
## 3    Arizona    6392017              NA
## 4   Arkansas    2915918              NA
## 5 California   37253956              55
## 6   Colorado    5029196              NA
```

```r
tab1 %>% right_join(tab2)
```

```
## Joining, by = "state"
```

```
##        state population electoral_votes
## 1 California   37253956              55
## 2      Texas         NA              38
## 3    Florida         NA              29
## 4   Illinois         NA              20
## 5       Ohio         NA              18
## 6    Georgia         NA              16
```

```r
inner_join(tab1, tab2)
```

```
## Joining, by = "state"
```

```
##        state population electoral_votes
## 1 California   37253956              55
```

```r
semi_join(tab1, tab2)
```

```
## Joining, by = "state"
```

```
##        state population
## 1 California   37253956
```

```r
anti_join(tab1, tab2)
```

```
## Joining, by = "state"
```

```
##      state population
## 1  Alabama    4779736
## 2   Alaska     710231
## 3  Arizona    6392017
## 4 Arkansas    2915918
## 5 Colorado    5029196
```

Binding

The textbook for this section is available here.

Key points

- Unlike the join functions, the binding functions do not try to match by a variable, but rather just combine datasets.
- `bind_cols()` binds two objects by making them columns in a tibble. The R-base function `cbind()` binds columns but makes a data frame or matrix instead.
- The `bind_rows()` function is similar but binds rows instead of columns. The R-base function `rbind()` binds rows but makes a data frame or matrix instead.

Code

```r
bind_cols(a = 1:3, b = 4:6)
```

```
## # A tibble: 3 x 2
##       a     b
##   <int> <int>
## 1     1     4
## 2     2     5
## 3     3     6
```

```r
tab1 <- tab[, 1:3]
tab2 <- tab[, 4:6]
tab3 <- tab[, 7:9]
new_tab <- bind_cols(tab1, tab2, tab3)
head(new_tab)
```

```
##        state abb region population total electoral_votes clinton trump others
## 1    Alabama  AL  South    4779736   135               9    34.4  62.1    3.6
## 2     Alaska  AK   West     710231    19               3    36.6  51.3   12.2
## 3    Arizona  AZ   West    6392017   232              11    45.1  48.7    6.2
## 4   Arkansas  AR  South    2915918    93               6    33.7  60.6    5.8
## 5 California  CA   West   37253956  1257              55    61.7  31.6    6.7
## 6   Colorado  CO   West    5029196    65               9    48.2  43.3    8.6
```

```r
tab1 <- tab[1:2,]
tab2 <- tab[3:4,]
bind_rows(tab1, tab2)
```

```
##      state abb region population total electoral_votes clinton trump others
## 1  Alabama  AL  South    4779736   135               9    34.4  62.1    3.6
## 2   Alaska  AK   West     710231    19               3    36.6  51.3   12.2
## 3  Arizona  AZ   West    6392017   232              11    45.1  48.7    6.2
## 4 Arkansas  AR  South    2915918    93               6    33.7  60.6    5.8
```

Set Operators

The textbook for this section is available here.

Key points

- By default, the set operators in R-base work on vectors. If **tidyverse/dplyr** are loaded, they also work on data frames.
- You can take intersections of vectors using `intersect()`. This returns the elements common to both sets.
- You can take the union of vectors using `union()`. This returns the elements that are in either set.
- The set difference between a first and second argument can be obtained with `setdiff()`. Note that this function is not symmetric.
- The function `set_equal()` tells us if two sets are the same, regardless of the order of elements.

Code

```r
# intersect vectors or data frames
intersect(1:10, 6:15)
```

```
## [1]  6  7  8  9 10
```

```r
intersect(c("a","b","c"), c("b","c","d"))
```

```
## [1] "b" "c"
```

```r
tab1 <- tab[1:5,]
tab2 <- tab[3:7,]
intersect(tab1, tab2)
```

```
##        state abb region population total electoral_votes clinton trump others
## 1    Arizona  AZ   West    6392017   232              11    45.1  48.7    6.2
## 2   Arkansas  AR  South    2915918    93               6    33.7  60.6    5.8
## 3 California  CA   West   37253956  1257              55    61.7  31.6    6.7
```

```r
# perform a union of vectors or data frames
union(1:10, 6:15)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

```r
union(c("a","b","c"), c("b","c","d"))
```

```
## [1] "a" "b" "c" "d"
```

```r
tab1 <- tab[1:5,]
tab2 <- tab[3:7,]
union(tab1, tab2)
```

```
##         state abb    region population total electoral_votes clinton trump
## 1     Alabama  AL     South    4779736   135               9    34.4  62.1
## 2      Alaska  AK      West     710231    19               3    36.6  51.3
## 3     Arizona  AZ      West    6392017   232              11    45.1  48.7
## 4    Arkansas  AR     South    2915918    93               6    33.7  60.6
## 5  California  CA      West   37253956  1257              55    61.7  31.6
## 6    Colorado  CO      West    5029196    65               9    48.2  43.3
## 7 Connecticut  CT Northeast    3574097    97               7    54.6  40.9
##   others
## 1    3.6
## 2   12.2
## 3    6.2
## 4    5.8
## 5    6.7
## 6    8.6
## 7    4.5
```

```r
# set difference of vectors or data frames
setdiff(1:10, 6:15)
```

```
## [1] 1 2 3 4 5
```

```r
setdiff(6:15, 1:10)
```

```
## [1] 11 12 13 14 15
```

```r
tab1 <- tab[1:5,]
tab2 <- tab[3:7,]
setdiff(tab1, tab2)
```

```
##     state abb region population total electoral_votes clinton trump others
## 1 Alabama  AL  South    4779736   135               9    34.4  62.1    3.6
## 2  Alaska  AK   West     710231    19               3    36.6  51.3   12.2
```

```r
# setequal determines whether sets have the same elements, regardless of order
setequal(1:5, 1:6)
```

```
## [1] FALSE
```

```r
setequal(1:5, 5:1)
```

```
## [1] TRUE
```

```r
setequal(tab1, tab2)
```

```
## [1] FALSE
```

Assessment - Combining Tables

1. You have created a `tab1` and `tab2` of state population and election data:

```r
> tab1
state                       population
Alabama                     4779736
Alaska                      710231
Arizona                     6392017
Delaware                    897934
District of Columbia        601723

> tab2
state          electoral_votes
Alabama        9
Alaska         3
Arizona        11
California     55
Colorado       9
Connecticut    7

> dim(tab1)
[1] 5 2

> dim(tab2)
[1] 6 2
```

What are the dimensions of the table dat, created by the following command?

```r
dat <- left_join(tab1, tab2, by = "state")
```

- ☐ A. 3 rows by 3 columns
- ☐ B. 5 rows by 2 columns
- ☒ C. 5 rows by 3 columns
- ☐ D. 6 rows by 3 columns

2. We are still using the `tab1` and "'tab2 tables shown in question 1. What join command would create a new table "dat" with three rows and two columns?

- ☐ A. `dat <- right_join(tab1, tab2, by = "state")`
- ☐ B. `dat <- full_join(tab1, tab2, by = "state")`
- ☐ C. `dat <- inner_join(tab1, tab2, by = "state")`

☒ D. `dat <- semi_join(tab1, tab2, by = "state")`

3. Which of the following are real differences between the join and bind functions?

☒ A. Binding functions combine by position, while join functions match by variables.
☒ B. Joining functions can join datasets of different dimensions, but the bind functions must match on the appropriate dimension (either same row or column numbers).
☒ C. Bind functions can combine both vectors and dataframes, while join functions work for only for dataframes.
☐ D. The join functions are a part of the dplyr package and have been optimized for speed, while the bind functions are inefficient base functions.

4. We have two simple tables, shown below, with columns `x` and `y`:

```
> df1
 x     y
 a     a
 b     a

> df2
 x     y
 a     a
 a     b
```

Which command would result in the following table?

```
> final
 x     y
 b     a
```

☐ A. `final <- union(df1, df2)`
☒ B. `final <- setdiff(df1, df2)`
☐ C. `final <- setdiff(df2, df1)`
☐ D. `final <- intersect(df1, df2)`

Install and load the **Lahman** library. This library contains a variety of datasets related to US professional baseball. We will use this library for the next few questions and will discuss it more extensively in the Regression course. For now, focus on wrangling the data rather than understanding the statistics.

The `Batting` data frame contains the offensive statistics for all baseball players over several seasons. Filter this data frame to define `top` as the top 10 home run (`HR`) hitters in 2016:

```
if(!require(Lahman)) install.packages("Lahman")
```

```
## Loading required package: Lahman
```

```
library(Lahman)
top <- Batting %>%
  filter(yearID == 2016) %>%
  arrange(desc(HR)) %>%    # arrange by descending HR count
  slice(1:10)    # take entries 1-10
top %>% as_tibble()
```

```
## # A tibble: 10 x 22
##    playerID yearID stint teamID lgID      G    AB     R     H   X2B   X3B    HR
##    <chr>     <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int> <int>
##  1 trumbma~   2016     1 BAL    AL      159   613    94   157    27     1    47
##  2 cruzne02   2016     1 SEA    AL      155   589    96   169    27     1    43
##  3 daviskh~   2016     1 OAK    AL      150   555    85   137    24     2    42
##  4 doziebr~   2016     1 MIN    AL      155   615   104   165    35     5    42
##  5 encared~   2016     1 TOR    AL      160   601    99   158    34     0    42
##  6 arenano~   2016     1 COL    NL      160   618   116   182    35     6    41
##  7 cartech~   2016     1 MIL    NL      160   549    84   122    27     1    41
##  8 frazito~   2016     1 CHA    AL      158   590    89   133    21     0    40
##  9 bryankr~   2016     1 CHN    NL      155   603   121   176    35     3    39
## 10 canoro01   2016     1 SEA    AL      161   655   107   195    33     2    39
## # ... with 10 more variables: RBI <int>, SB <int>, CS <int>, BB <int>,
## #   SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>, GIDP <int>
```

Also Inspect the `Master` data frame, which has demographic information for all players:

```
Master %>% as_tibble()
```

```
## # A tibble: 19,878 x 26
##    playerID birthYear birthMonth birthDay birthCountry birthState birthCity
##    <chr>        <int>      <int>    <int> <chr>        <chr>      <chr>
##  1 aardsda~      1981         12       27 USA          CO         Denver
##  2 aaronha~      1934          2        5 USA          AL         Mobile
##  3 aaronto~      1939          8        5 USA          AL         Mobile
##  4 aasedo01      1954          9        8 USA          CA         Orange
##  5 abadan01      1972          8       25 USA          FL         Palm Bea~
##  6 abadfe01      1985         12       17 D.R.         La Romana  La Romana
##  7 abadijo~      1850         11        4 USA          PA         Philadel~
##  8 abbated~      1877          4       15 USA          PA         Latrobe
##  9 abbeybe~      1869         11       11 USA          VT         Essex
## 10 abbeych~      1866         10       14 USA          NE         Falls Ci~
## # ... with 19,868 more rows, and 19 more variables: deathYear <int>,
## #   deathMonth <int>, deathDay <int>, deathCountry <chr>, deathState <chr>,
## #   deathCity <chr>, nameFirst <chr>, nameLast <chr>, nameGiven <chr>,
## #   weight <int>, height <int>, bats <fct>, throws <fct>, debut <chr>,
## #   finalGame <chr>, retroID <chr>, bbrefID <chr>, deathDate <date>,
## #   birthDate <date>
```

5. Use the correct `join` or `bind` function to create a combined table of the names and statistics of the top 10 home run (HR) hitters for 2016. This table should have the player ID, first name, last name, and number of HR for the top 10 players. Name this data frame `top_names`.

Identify the `join` or `bind` that fills the blank in this code to create the correct table:

```
top_names <- top %>% ___________________ %>%
    select(playerID, nameFirst, nameLast, HR)
```

Which bind or join function fills the blank to generate the correct table?

```r
top_names <- top %>% left_join(Master) %>%
    select(playerID, nameFirst, nameLast, HR)
```

```
## Joining, by = "playerID"
```

- ☐ A. rbind(Master)
- ☐ B. cbind(Master)
- ☒ C. left_join(Master)
- ☐ D. right_join(Master)
- ☐ E. full_join(Master)
- ☐ F. anti_join(Master)

6. Inspect the `Salaries` data frame. Filter this data frame to the 2016 salaries, then use the correct bind join function to add a `salary` column to the `top_names` data frame from the previous question. Name the new data frame `top_salary`. Use this code framework:

```r
top_salary <- Salaries %>% filter(yearID == 2016) %>%
  _______________ %>%
  select(nameFirst, nameLast, teamID, HR, salary)
```

Which `bind` or `join` function fills the blank to generate the correct table?

```r
top_salary <- Salaries %>% filter(yearID == 2016) %>%
  right_join(top_names) %>%
  select(nameFirst, nameLast, teamID, HR, salary)
```

- ☐ A. rbind(top_names)
- ☐ B. cbind(top_names)
- ☐ C. left_join(top_names)
- ☒ D. right_join(top_names)
- ☐ E. full_join(top_names)
- ☐ F. anti_join(top_names)

7. Inspect the `AwardsPlayers` table. Filter awards to include only the year 2016.

How many players from the top 10 home run hitters won at least one award in 2016? Use a set operator.

```r
Awards_2016 <- AwardsPlayers %>% filter(yearID == 2016)
length(intersect(Awards_2016$playerID, top_names$playerID))
```

```
## [1] 3
```

How many players won an award in 2016 but were not one of the top 10 home run hitters in 2016? Use a set operator.

```r
length(setdiff(Awards_2016$playerID, top_names$playerID))
```

```
## [1] 44
```

Web Scraping

The textbook for this section is available here through section 23.2.

Key points

- Web scraping is extracting data from a website.
- The **rvest** web harvesting package includes functions to extract nodes of an HTML document: `html_nodes()` extracts all nodes of different types, and `html_node()` extracts the first node.
- `html_table()` converts an HTML table to a data frame.

Code

```
# import a webpage into R
if(!require(rvest)) install.packages("rvest")
```

```
## Loading required package: rvest

## Loading required package: xml2

##
## Attaching package: 'rvest'

## The following object is masked from 'package:purrr':
##
##     pluck

## The following object is masked from 'package:readr':
##
##     guess_encoding
```

```
library(rvest)
url <- "https://en.wikipedia.org/wiki/Murder_in_the_United_States_by_state"
h <- read_html(url)
class(h)
```

```
## [1] "xml_document" "xml_node"
```

```
h
```

```
## {html_document}
## <html class="client-nojs" lang="en" dir="ltr">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
## [2] <body class="mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-0 ns-subject  ...
```

```
tab <- h %>% html_nodes("table")
tab <- tab[[2]]

tab <- tab %>% html_table
class(tab)
```

```
## [1] "data.frame"
```

```r
tab <- tab %>% setNames(c("state", "population", "total", "murders", "gun_murders", "gun_ownership", "to
head(tab)
```

```
##        state population total murders gun_murders gun_ownership total_rate
## 1    Alabama  4,853,875   348    -[a]        -[a]          48.9        7.2
## 2     Alaska    737,709    59      57          39          61.7        8.0
## 3    Arizona  6,817,565   306     278         171          32.3        4.5
## 4   Arkansas  2,977,853   181     164         110          57.9        6.1
## 5 California 38,993,940 1,861   1,861       1,275          20.1        4.8
## 6   Colorado  5,448,819   176     176         115          34.3        3.2
##   murder_rate gun_murder_rate
## 1       - [a]           - [a]
## 2         7.7             5.3
## 3         4.1             2.5
## 4         5.5             3.7
## 5         4.8             3.3
## 6         3.2             2.1
```

CSS Selectors

This page corresponds to the textbook section on CSS selectors.

The default look of webpages made with the most basic HTML is quite unattractive. The aesthetically pleasing pages we see today are made using CSS. CSS is used to add style to webpages. The fact that all pages for a company have the same style is usually a result that they all use the same CSS file. The general way these CSS files work is by defining how each of the elements of a webpage will look. The title, headings, itemized lists, tables, and links, for example, each receive their own style including font, color, size, and distance from the margin, among others.

To do this CSS leverages patterns used to define these elements, referred to as selectors. An example of pattern we used in a previous video is table but there are many many more. If we want to grab data from a webpage and we happen to know a selector that is unique to the part of the page, we can use the html_nodes() function.

However, knowing which selector to use can be quite complicated. To demonstrate this we will try to extract the recipe name, total preparation time, and list of ingredients from this guacamole recipe. Looking at the code for this page, it seems that the task is impossibly complex. However, selector gadgets actually make this possible. SelectorGadget is piece of software that allows you to interactively determine what CSS selector you need to extract specific components from the webpage. If you plan on scraping data other than tables, we highly recommend you install it. A Chrome extension is available which permits you to turn on the gadget highlighting parts of the page as you click through, showing the necessary selector to extract those segments.

For the guacamole recipe page, we already have done this and determined that we need the following selectors:

```r
h <- read_html("http://www.foodnetwork.com/recipes/alton-brown/guacamole-recipe-1940609")
recipe <- h %>% html_node(".o-AssetTitle__a-HeadlineText") %>% html_text()
prep_time <- h %>% html_node(".m-RecipeInfo__a-Description--Total") %>% html_text()
ingredients <- h %>% html_nodes(".o-Ingredients__a-Ingredient") %>% html_text()
```

You can see how complex the selectors are. In any case we are now ready to extract what we want and create a list:

```
guacamole <- list(recipe, prep_time, ingredients)
guacamole
```

Since recipe pages from this website follow this general layout, we can use this code to create a function that extracts this information:

```
get_recipe <- function(url){
    h <- read_html(url)
    recipe <- h %>% html_node(".o-AssetTitle__a-HeadlineText") %>% html_text()
    prep_time <- h %>% html_node(".m-RecipeInfo__a-Description--Total") %>% html_text()
    ingredients <- h %>% html_nodes(".o-Ingredients__a-Ingredient") %>% html_text()
    return(list(recipe = recipe, prep_time = prep_time, ingredients = ingredients))
}
```

and then use it on any of their webpages:

```
get_recipe("http://www.foodnetwork.com/recipes/food-network-kitchen/pancakes-recipe-1913844")
```

There are several other powerful tools provided by **rvest**. For example, the functions `html_form()`, `set_values()`, and `submit_form()` permit you to query a webpage from R. This is a more advanced topic not covered here.

Assessment - Web Scraping

Load the following web page, which contains information about Major League Baseball payrolls, into R:

https://web.archive.org/web/20181024132313/http://www.stevetheump.com/Payrolls.htm

```
url <- "https://web.archive.org/web/20181024132313/http://www.stevetheump.com/Payrolls.htm"
h <- read_html(url)
```

We learned that tables in html are associated with the `table` node. Use the `html_nodes()` function and the `table` node type to extract the first table. Store it in an object nodes:

```
nodes <- html_nodes(h, "table")
```

The `html_nodes()` function returns a list of objects of class `xml_node`. We can see the content of each one using, for example, the `html_text()` function. You can see the content for an arbitrarily picked component like this:

```
html_text(nodes[[8]])
```

```
## [1] "Team\nPayroll\nAverge\nMedianNew York Yankees\n$ 197,962,289\n$ 6,186,321\n$ 1,937,500Philadelph
```

If the content of this object is an html table, we can use the `html_table()` function to convert it to a data frame:

```
html_table(nodes[[8]])
```

```
##                     Team       Payroll      Averge      Median
## 1        New York Yankees $ 197,962,289 $ 6,186,321 $ 1,937,500
## 2   Philadelphia Phillies $ 174,538,938 $ 5,817,964 $ 1,875,000
## 3          Boston Red Sox $ 173,186,617 $ 5,093,724 $ 1,556,250
## 4      Los Angeles Angels $ 154,485,166 $ 5,327,074 $ 3,150,000
## 5          Detroit Tigers $ 132,300,000 $ 4,562,068 $ 1,100,000
## 6           Texas Rangers $ 120,510,974 $ 4,635,037 $ 3,437,500
## 7           Miami Marlins $ 118,078,000 $ 4,373,259 $ 1,500,000
## 8    San Francisco Giants $ 117,620,683 $ 3,920,689 $ 1,275,000
## 9     St. Louis Cardinals $ 110,300,862 $ 3,939,316   $ 800,000
## 10      Milwaukee Brewers  $ 97,653,944 $ 3,755,920 $ 1,981,250
## 11      Chicago White Sox  $ 96,919,500 $ 3,876,780   $ 530,000
## 12    Los Angeles Dodgers  $ 95,143,575 $ 3,171,452   $ 875,000
## 13        Minnesota Twins  $ 94,085,000 $ 3,484,629   $ 750,000
## 14          New York Mets  $ 93,353,983 $ 3,457,554   $ 875,000
## 15           Chicago Cubs  $ 88,197,033 $ 3,392,193 $ 1,262,500
## 16         Atlanta Braves  $ 83,309,942 $ 2,776,998   $ 577,500
## 17        Cincinnati Reds  $ 82,203,616 $ 2,935,843 $ 1,150,000
## 18       Seattle Mariners  $ 81,978,100 $ 2,927,789   $ 495,150
## 19      Baltimore Orioles  $ 81,428,999 $ 2,807,896 $ 1,300,000
## 20   Washington Nationals  $ 81,336,143 $ 2,623,746   $ 800,000
## 21      Cleveland Indians  $ 78,430,300 $ 2,704,493   $ 800,000
## 22       Colorado Rockies  $ 78,069,571 $ 2,692,054   $ 482,000
## 23      Toronto Blue Jays  $ 75,489,200 $ 2,696,042 $ 1,768,750
## 24   Arizona Diamondbacks  $ 74,284,833 $ 2,653,029 $ 1,625,000
## 25          Tampa Bay Rays  $ 64,173,500 $ 2,291,910 $ 1,425,000
## 26     Pittsburgh Pirates  $ 63,431,999 $ 2,187,310   $ 916,666
## 27     Kansas City Royals  $ 60,916,225 $ 2,030,540   $ 870,000
## 28         Houston Astros  $ 60,651,000 $ 2,332,730   $ 491,250
## 29      Oakland Athletics  $ 55,372,500 $ 1,845,750   $ 487,500
## 30       San Diego Padres  $ 55,244,700 $ 1,973,025 $ 1,207,500
```

You will analyze the tables from this HTML page over questions 1-3.

1. Many tables on this page are team payroll tables, with columns for rank, team, and one or more money values.

Convert the first four tables in `nodes` to data frames and inspect them.

```r
sapply(nodes[1:4], html_table)    # 2, 3, 4 give tables with payroll info
```

```
## [[1]]
##   X1                                                             X2
## 1 NA Salary Stats 1967-2019\nTop ML Player Salaries / Baseball's Luxury Tax
##
## [[2]]
##    RANK                  TEAM   Payroll
## 1     1        Boston Red Sox  $235.65M
## 2     2  San Francisco Giants  $208.51M
## 3     3   Los Angeles Dodgers  $186.14M
## 4     4          Chicago Cubs  $183.46M
## 5     5  Washington Nationals  $181.59M
## 6     6    Los Angeles Angels   $175.1M
```

```
## 7     7      New York Yankees  $168.54M
## 8     8      Seattle Mariners  $162.48M
## 9     9     Toronto Blue Jays $162.316M
## 10   10   St. Louis Cardinals  $161.01M
## 11   11        Houston Astros  $160.04M
## 12   12         New York Mets  $154.61M
## 13   13         Texas Rangers   $144.0M
## 14   14     Baltimore Orioles  $143.09M
## 15   15      Colorado Rockies  $141.34M
## 16   16     Cleveland Indians  $134.35M
## 17   17  Arizona Diamondbacks   $132.5M
## 18   18       Minnesota Twins  $131.91M
## 19   19        Detroit Tigers  $129.92M
## 20   20    Kansas City Royals  $129.92M
## 21   21        Atlanta Braves  $120.54M
## 22   22       Cincinnati Reds  $101.19M
## 23   23         Miami Marlins   $98.64M
## 24   24 Philadelphia Phillies   $96.85M
## 25   25      San Diego Padres   $96.13M
## 26   26     Milwaukee Brewers   $90.24M
## 27   27    Pittsburgh Pirates   $87.88M
## 28   28        Tampa Bay Rays   $78.73M
## 29   29     Chicago White Sox   $72.18M
## 30   30     Oakland Athletics   $68.53M
##
## [[3]]
##      X1                           X2           X3           X4            X5
## 1  Rank                         Team       25 Man Disabled List Total Payroll
## 2     1          Los Angeles Dodgers $155,887,854  $37,354,166  $242,065,828
## 3     2             New York Yankees $168,045,699   $5,644,000  $201,539,699
## 4     3               Boston Red Sox $136,780,500  $38,239,250  $199,805,178
## 5     4               Detroit Tigers $168,500,600  $11,750,000  $199,750,600
## 6     5            Toronto Blue Jays $159,175,968   $2,169,400  $177,795,368
## 7     6               Texas Rangers $115,162,703  $39,136,360  $175,909,063
## 8     7         San Francisco Giants $169,504,611   $2,500,000  $172,354,611
## 9     8                 Chicago Cubs $170,189,880   $2,000,000  $172,189,880
## 10    9         Washington Nationals $163,111,918     $535,000  $167,846,918
## 11   10            Baltimore Orioles $142,066,615  $19,501,668  $163,676,616
## 12   11 Los Angeles Angels of Anaheim $116,844,833  $17,120,500  $160,375,333
## 13   12                New York Mets $120,870,470  $26,141,990  $155,187,460
## 14   13             Seattle Mariners $139,257,018  $15,007,300  $154,800,918
## 15   14          St. Louis Cardinals $136,181,533  $13,521,400  $152,452,933
## 16   15           Kansas City Royals $127,333,150   $4,092,100  $140,925,250
## 17   16             Colorado Rockies  $86,909,571  $14,454,000  $130,963,571
## 18   17            Cleveland Indians $101,105,399  $14,005,766  $124,861,165
## 19   18               Houston Astros $117,957,800   $4,386,100  $124,343,900
## 20   19               Atlanta Braves $103,303,791   $8,927,500  $112,437,541
## 21   20                Miami Marlins  $96,446,100  $15,035,000  $111,881,100
## 22   21        Philadelphia Phillies  $86,841,000     $537,000  $111,378,000
## 23   22              Minnesota Twins  $92,592,500   $8,735,000  $108,077,500
## 24   23           Pittsburgh Pirates  $92,362,832            -  $100,575,946
## 25   24            Chicago White Sox  $95,625,000   $1,671,000   $99,119,770
## 26   25              Cincinnati Reds  $53,858,785  $26,910,000   $93,768,785
## 27   26         Arizona Diamondbacks  $91,481,600   $1,626,000   $93,257,600
```

```
## 28   27             Oakland Athletics  $64,339,166    $5,732,500   $81,738,333
## 29   28              San Diego Padres  $29,628,400    $4,946,000   $71,624,400
## 30   29                Tampa Bay Rays  $55,282,232   $14,680,300   $69,962,532
## 31   30             Milwaukee Brewers  $50,023,900   $13,037,400   $63,061,300
##
## [[4]]
##    Rank          Team  Opening Day  Avg Salary      Median
## 1     1       Dodgers $ 223,352,402 $ 7,445,080 $ 5,166,666
## 2     2       Yankees $ 213,472,857 $ 7,361,133 $ 3,300,000
## 3     3       Red Sox $ 182,161,414 $ 6,072,047 $ 3,500,000
## 4     4        Tigers $ 172,282,250 $ 6,891,290 $ 3,000,000
## 5     5        Giants $ 166,495,942 $ 5,946,284 $ 4,000,000
## 6     6     Nationals $ 166,010,977 $ 5,724,516 $ 2,500,000
## 7     7        Angels $ 146,449,583 $ 5,049,986 $ 1,312,500
## 8     8       Rangers $ 144,307,373 $ 4,509,605   $ 937,500
## 9     9      Phillies $ 133,048,000 $ 4,434,933   $ 700,000
## 10   10     Blue Jays $ 126,369,628 $ 4,357,573 $ 1,650,000
## 11   11      Mariners $ 122,706,842 $ 4,719,494 $ 2,252,500
## 12   12     Cardinals $ 120,301,957 $ 4,455,628 $ 2,000,000
## 13   13          Reds $ 116,732,284 $ 4,323,418 $ 2,350,000
## 14   14          Cubs $ 116,654,522 $ 4,166,233 $ 2,515,000
## 15   15       Orioles $ 115,587,632 $ 3,985,780 $ 2,750,000
## 16   16        Royals $ 112,914,525 $ 4,032,662 $ 2,532,500
## 17   17        Padres $ 112,895,700 $ 4,342,142   $ 763,500
## 18   18         Twins $ 108,262,000 $ 4,163,923 $ 1,775,000
## 19   19          Mets  $ 99,626,453 $ 3,558,088   $ 669,562
## 20   20     White Sox  $ 98,712,867 $ 3,525,460 $ 1,250,000
## 21   21       Brewers  $ 98,683,035 $ 3,795,501   $ 529,750
## 22   22       Rockies  $ 98,261,171 $ 3,388,316 $ 1,087,600
## 23   23        Braves  $ 87,622,648 $ 2,920,755 $ 1,333,333
## 24   24       Indians  $ 86,339,067 $ 3,197,743 $ 1,940,000
## 25   25       Pirates  $ 85,885,832 $ 2,862,861 $ 1,279,166
## 26   26       Marlins  $ 84,637,500 $ 3,134,722 $ 1,925,000
## 27   27     Athletics  $ 80,279,166 $ 2,508,724   $ 648,750
## 28   28          Rays  $ 73,649,584 $ 2,454,986   $ 750,000
## 29   29 Diamondbacks  $ 70,762,833 $ 2,358,761   $ 663,000
## 30   30        Astros  $ 69,064,200 $ 2,466,579 $ 1,031,250
```

Which of the first four `nodes` are tables of team payroll? Check all correct answers. Look at table content, not column names.

- ☐ A. None of the above
- ☐ B. Table 1
- ☒ C. Table 2
- ☒ D. Table 3
- ☒ E. Table 4

2. For the last 3 components of `nodes`, which of the following are true? Check all correct answers.

```r
html_table(nodes[[length(nodes)-2]])
```

```
##                    X1           X2         X3
## 1                Team      Payroll    Average
```

```
## 2         NY Yankees $109,791,893 $3,541,674
## 3             Boston $109,558,908 $3,423,716
## 4        Los Angeles $108,980,952 $3,757,964
## 5            NY Mets  $93,174,428 $3,327,658
## 6          Cleveland  $91,974,979 $3,065,833
## 7            Atlanta  $91,851,687 $2,962,958
## 8              Texas  $88,504,421 $2,854,981
## 9            Arizona  $81,206,513 $2,900,233
## 10         St. Louis  $77,270,855 $2,664,512
## 11           Toronto  $75,798,500 $2,707,089
## 12           Seattle  $75,652,500 $2,701,875
## 13         Baltimore  $72,426,328 $2,497,460
## 14          Colorado  $71,068,000 $2,632,148
## 15      Chicago Cubs  $64,015,833 $2,462,147
## 16     San Francisco  $63,332,667 $2,345,654
## 17 Chicago White Sox  $62,363,000 $2,309,741
## 18           Houston  $60,382,667 $2,236,395
## 19         Tampa Bay  $54,951,602 $2,035,245
## 20        Pittsburgh  $52,698,333 $1,699,946
## 21           Detroit  $49,831,167 $1,779,685
## 22           Anaheim  $46,568,180 $1,502,199
## 23        Cincinnati  $45,227,882 $1,739,534
## 24         Milwaukee  $43,089,333 $1,595,901
## 25      Philadelphia  $41,664,167 $1,602,468
## 26         San Diego  $38,333,117 $1,419,745
## 27       Kansas City  $35,643,000 $1,229,069
## 28           Florida  $35,504,167 $1,183,472
## 29          Montreal  $34,774,500 $1,159,150
## 30           Oakland  $33,810,750 $1,252,250
## 31         Minnesota  $24,350,000   $901,852
```

```r
html_table(nodes[[length(nodes)-1]])
```

```
##                   X1          X2         X3
## 1               Team     Payroll    Average
## 2         NY Yankees $92,538,260 $3,190,974
## 3        Los Angeles $88,124,286 $3,263,862
## 4            Atlanta $84,537,836 $2,817,928
## 5          Baltimore $81,447,435 $2,808,532
## 6            Arizona $81,027,833 $2,893,851
## 7            NY Mets $79,509,776 $3,180,391
## 8             Boston $77,940,333 $2,598,011
## 9          Cleveland $75,880,871 $2,918,495
## 10             Texas $70,795,921 $2,722,920
## 11         Tampa Bay $62,765,129 $2,024,682
## 12         St. Louis $61,453,863 $2,276,069
## 13          Colorado $61,111,190 $2,182,543
## 14      Chicago Cubs $60,539,333 $2,017,978
## 15           Seattle $58,915,000 $2,265,962
## 16           Detroit $58,265,167 $2,157,969
## 17         San Diego $54,821,000 $1,827,367
## 18     San Francisco $53,737,826 $2,066,839
## 19           Anaheim $51,464,167 $1,715,472
## 20           Houston $51,289,111 $1,899,597
```

```
## 21      Philadelphia $47,308,000 $1,631,310
## 22        Cincinnati $46,867,200 $1,735,822
## 23           Toronto $46,238,333 $1,778,397
## 24         Milwaukee $36,505,333 $1,140,792
## 25          Montreal $34,807,833 $1,200,270
## 26           Oakland $31,971,333 $1,184,123
## 27 Chicago White Sox $31,133,500 $1,073,569
## 28        Pittsburgh $28,928,333 $1,112,628
## 29       Kansas City $23,433,000   $836,893
## 30           Florida $20,072,000   $692,138
## 31         Minnesota $16,519,500   $635,365
```

```r
html_table(nodes[[length(nodes)]])
```

```
##      X1       X2         X3     X4
## 1  Year  Minimum    Average  % Chg
## 2  2019 $555,000                 -
## 3  2018 $545,000 $4,520,000
## 4  2017 $535,000 $4,470,000    5.4
## 5  2016 $507,500 $4,400,000      -
## 6  2015 $507,500 $4,250,000      -
## 7  2014 $507,500 $3,820,000   12.8
## 8  2013 $480,000 $3,386,212    5.4
## 9  2012 $480,000 $3,440,000    3.8
## 10 2011 $414,500 $3,305,393    0.2
## 11 2010 $400,000 $3,297,828    1.8
## 12 2009 $400,000 $3,240,206    2.7
## 13 2008 $390,000 $3,150,000    7.1
## 14 2007 $380,000 $2,820,000    4.6
## 15 2006 $327,000 $2,699,292      9
## 16 2005 $316,000 $2,632,655    5.9
## 17 2004 $300,000 $2,486,609 (-2.7)
## 18 2003 $300,000 $2,555,416    7.2
## 19 2002 $200,000 $2,340,920    5.2
## 20 2001 $200,000 $2,138,896   13.9
## 21 2000 $200,000 $1,895,630   15.6
## 22 1999 $200,000 $1,611,166   19.3
## 23 1998 $170,000 $1,398,831    4.2
## 24 1997 $150,000 $1,336,609   17.6
## 25 1996 $122,667 $1,119,981    9.9
## 26 1995 $109,000 $1,110,766 (-9.9)
## 27 1994 $109,000 $1,168,263    6.1
## 28 1993 $109,000 $1,076,089    3.3
## 29 1992 $109,000 $1,028,667   21.7
## 30 1991 $100,000   $851,492   53.9
## 31 1990 $100,000   $597,537   12.9
## 32 1989  $68,000   $497,254
## 33 1988  $62,500   $438,729
## 34 1987  $62,500   $412,454
## 35 1986  $60,000   $412,520
## 36 1985  $60,000   $371,571
## 37 1984  $40,000   $329,408
## 38 1983  $35,000   $289,194
## 39 1982  $33,500   $241,497
```

```
## 40 1981  $32,500   $185,651
## 41 1980  $30,000   $143,756
## 42 1979  $21,000   $113,558
## 43 1978  $21,000    $99,876
## 44 1977  $19,000    $76,066
## 45 1976  $19,000    $51,501
## 46 1975  $16,000    $44,676
## 47 1974  $15,000    $40,839
## 48 1973  $15,000    $36,566
## 49 1972  $13,500    $34,092
## 50 1971  $12,750    $31,543
## 51 1970  $12,000    $29,303
## 52 1969  $10,000    $24,909
## 53 1968  $10,000        N/A
## 54 1967   $6,000    $19,000
```

- ☒ A. All three entries are tables.
- ☐ B. All three entries are tables of payroll per team.
- ☒ C. The last entry shows the average across all teams through time, not payroll per team.
- ☐ D. None of the three entries are tables of payroll per team.

3. Create a table called `tab_1` using entry 10 of `nodes`. Create a table called `tab_2` using entry 19 of `nodes`.

Note that the column names should be `c("Team", "Payroll", "Average")`. You can see that these column names are actually in the first data row of each table, and that `tab_1` has an extra first column `No.` that should be removed so that the column names for both tables match.

Remove the extra column in `tab_1`, remove the first row of each dataset, and change the column names for each table to `c("Team", "Payroll", "Average")`. Use a `full_join()` by the `Team` to combine these two tables.

Note that some students, presumably because of system differences, have noticed that entry 18 instead of entry 19 of `nodes` gives them the `tab_2` correctly; be sure to check entry 18 if entry 19 is giving you problems.

How many rows are in the joined data table?

```r
tab_1 <- html_table(nodes[[10]])
tab_2 <- html_table(nodes[[19]])
col_names <- c("Team", "Payroll", "Average")
tab_1 <- tab_1[-1, -1]
tab_2 <- tab_2[-1,]
names(tab_2) <- col_names
names(tab_1) <- col_names
full_join(tab_1,tab_2, by = "Team")
```

```
##                     Team    Payroll.x  Average.x   Payroll.y  Average.y
## 1       New York Yankees $206,333,389 $8,253,336        <NA>       <NA>
## 2         Boston Red Sox $162,747,333 $5,611,977        <NA>       <NA>
## 3           Chicago Cubs $146,859,000 $5,439,222 $64,015,833 $2,462,147
## 4  Philadelphia Phillies $141,927,381 $5,068,835        <NA>       <NA>
## 5          New York Mets $132,701,445 $5,103,902        <NA>       <NA>
## 6         Detroit Tigers $122,864,929 $4,550,553        <NA>       <NA>
## 7      Chicago White Sox $108,273,197 $4,164,354 $62,363,000 $2,309,741
```

```
## 8     Los Angeles Angels $105,013,667 $3,621,161         <NA>       <NA>
## 9       Seattle Mariners  $98,376,667 $3,513,452         <NA>       <NA>
## 10  San Francisco Giants  $97,828,833 $3,493,887         <NA>       <NA>
## 11       Minnesota Twins  $97,559,167 $3,484,256         <NA>       <NA>
## 12   Los Angeles Dodgers  $94,945,517 $3,651,751         <NA>       <NA>
## 13   St. Louis Cardinals  $93,540,753 $3,741,630         <NA>       <NA>
## 14        Houston Astros  $92,355,500 $3,298,411         <NA>       <NA>
## 15        Atlanta Braves  $84,423,667 $3,126,802         <NA>       <NA>
## 16      Colorado Rockies  $84,227,000 $2,904,379         <NA>       <NA>
## 17     Baltimore Orioles  $81,612,500 $3,138,942         <NA>       <NA>
## 18     Milwaukee Brewers  $81,108,279 $2,796,837         <NA>       <NA>
## 19       Cincinnati Reds  $72,386,544 $2,784,098         <NA>       <NA>
## 20    Kansas City Royals  $72,267,710 $2,491,990         <NA>       <NA>
## 21        Tampa Bay Rays  $71,923,471 $2,663,832         <NA>       <NA>
## 22     Toronto Blue Jays  $62,689,357 $2,089,645         <NA>       <NA>
## 23  Washington Nationals  $61,425,000 $2,047,500         <NA>       <NA>
## 24     Cleveland Indians  $61,203,967 $2,110,482         <NA>       <NA>
## 25  Arizona Diamondbacks  $60,718,167 $2,335,314         <NA>       <NA>
## 26       Florida Marlins  $55,641,500 $2,060,796         <NA>       <NA>
## 27         Texas Rangers  $55,250,545 $1,905,191         <NA>       <NA>
## 28     Oakland Athletics  $51,654,900 $1,666,287         <NA>       <NA>
## 29      San Diego Padres  $37,799,300 $1,453,819         <NA>       <NA>
## 30    Pittsburgh Pirates  $34,943,000 $1,294,185         <NA>       <NA>
## 31            NY Yankees         <NA>       <NA> $109,791,893 $3,541,674
## 32                Boston         <NA>       <NA> $109,558,908 $3,423,716
## 33           Los Angeles         <NA>       <NA> $108,980,952 $3,757,964
## 34               NY Mets         <NA>       <NA>  $93,174,428 $3,327,658
## 35             Cleveland         <NA>       <NA>  $91,974,979 $3,065,833
## 36               Atlanta         <NA>       <NA>  $91,851,687 $2,962,958
## 37                 Texas         <NA>       <NA>  $88,504,421 $2,854,981
## 38               Arizona         <NA>       <NA>  $81,206,513 $2,900,233
## 39             St. Louis         <NA>       <NA>  $77,270,855 $2,664,512
## 40               Toronto         <NA>       <NA>  $75,798,500 $2,707,089
## 41               Seattle         <NA>       <NA>  $75,652,500 $2,701,875
## 42             Baltimore         <NA>       <NA>  $72,426,328 $2,497,460
## 43              Colorado         <NA>       <NA>  $71,068,000 $2,632,148
## 44         San Francisco         <NA>       <NA>  $63,332,667 $2,345,654
## 45               Houston         <NA>       <NA>  $60,382,667 $2,236,395
## 46             Tampa Bay         <NA>       <NA>  $54,951,602 $2,035,245
## 47            Pittsburgh         <NA>       <NA>  $52,698,333 $1,699,946
## 48               Detroit         <NA>       <NA>  $49,831,167 $1,779,685
## 49               Anaheim         <NA>       <NA>  $46,568,180 $1,502,199
## 50            Cincinnati         <NA>       <NA>  $45,227,882 $1,739,534
## 51             Milwaukee         <NA>       <NA>  $43,089,333 $1,595,901
## 52          Philadelphia         <NA>       <NA>  $41,664,167 $1,602,468
## 53             San Diego         <NA>       <NA>  $38,333,117 $1,419,745
## 54           Kansas City         <NA>       <NA>  $35,643,000 $1,229,069
## 55               Florida         <NA>       <NA>  $35,504,167 $1,183,472
## 56              Montreal         <NA>       <NA>  $34,774,500 $1,159,150
## 57               Oakland         <NA>       <NA>  $33,810,750 $1,252,250
## 58             Minnesota         <NA>       <NA>  $24,350,000   $901,852
```

4. The Wikipedia page on opinion polling for the Brexit referendum, in which the United Kingdom voted to leave the European Union in June 2016, contains several tables. One table contains the results of

all polls regarding the referendum over 2016.

2016 [edit]

Date(s) conducted	Remain	Leave	Undecided	Lead	Sample	Conducted by	Polling type	Notes
23 June 2016	*48.1%*	***51.9%***	*N/A*	*3.8%*	*33,577,342*	*Results of the United Kingdom European Union membership referendum, 2016*	*UK-wide referendum*	
23 June	**52%**	48%	N/A	4%	4,772	YouGov	Online	On the day poll
22 June	**55%**	45%	N/A	10%	4,700	Populus	Online	
20–22 June	**51%**	49%	N/A	2%	3,766	YouGov	Online	Includes Northern Ireland (turnout weighted)
20–22 June	**49%**	46%	1%	3%	1,592	Ipsos MORI	Telephone	
20–22 June	44%	**45%**	9%	1%	3,011	Opinium	Online	
17–22 June	**54%**	46%	N/A	8%	1,032	ComRes	Telephone	Those expressing a voting intention (turnout weighted)
	48%	42%	11%	6%				All UK adults (turnout weighted)
16–22 June	41%	**43%**	16%	2%	2,320	TNS	Online	
20 June	**45%**	44%	11%	1%	1,003	Survation/IG Group	Telephone	
18–19 June	42%	**44%**	13%	2%	1,652	YouGov	Online	
16–19 June	**53%**	46%	2%	7%	800	ORB/Telegraph	Telephone	Definite voters only
17–18 June	**45%**	42%	13%	3%	1,004	Survation	Telephone	

Figure 1: Polls regarding the referendum over 2016

Use the rvest library to read the HTML from this Wikipedia page (make sure to copy both lines of the URL):

```
url <- "https://en.wikipedia.org/w/index.php?title=Opinion_polling_for_the_United_Kingdom_European_Unio
```

Assign `tab` to be the html nodes of the "table" class.

How many tables are in this Wikipedia page?

```
tab <- read_html(url) %>% html_nodes("table")
length(tab)
```

```
## [1] 39
```

5. Inspect the first several html tables using `html_table()` with the argument `fill=TRUE` (you can read about this argument in the documentation). Find the first table that has 9 columns with the first column named "Date(s) conducted".

What is the first table number to have 9 columns where the first column is named "Date(s) conducted"?

```
tab[[5]] %>% html_table(fill = TRUE) %>% names()     # inspect column names
```

```
## [1] "Date(s) conducted" "Remain"            "Leave"
## [4] "Undecided"         "Lead"              "Sample"
## [7] "Conducted by"      "Polling type"      "Notes"
```

Section 3 Overview

In the **String Processing** section, we use case studies that help demonstrate how string processing is a powerful tool useful for overcoming many data wrangling challenges. You will see how the original **raw** data was processed to create the data frames we have used in courses throughout this series.

This section is divided into three parts.

After completing the **String Processing** section, you will be able to:

- **Remove** unwanted characters from text.
- **Extract numeric** values from text.
- **Find** and **replace** characters.
- **Extract specific** parts of strings.
- **Convert** free form text into more uniform formats.
- **Split strings** into multiple values.
- Use **regular expressions (regex)** to process strings.

String Parsing

The textbook for this section is available here.

Key points

- The most common tasks in string processing include:
  - extracting numbers from strings
  - removing unwanted characters from text
  - finding and replacing characters
  - extracting specific parts of strings
  - converting free form text to more uniform formats
  - splitting strings into multiple values
- The **stringr** package in the **tidyverse** contains string processing functions that follow a similar naming format (`str_functionname`) and are compatible with the pipe.

Code

```
# read in raw murders data from Wikipedia
url <- "https://en.wikipedia.org/w/index.php?title=Gun_violence_in_the_United_States_by_state&direction=
murders_raw <- read_html(url) %>%
  html_nodes("table") %>%
  html_table() %>%
  .[[1]] %>%
  setNames(c("state", "population", "total", "murder_rate"))

# inspect data and column classes
head(murders_raw)
```

```
##        state population total murder_rate
## 1    Alabama  4,853,875   348         7.2
## 2     Alaska    737,709    59         8.0
## 3    Arizona  6,817,565   309         4.5
## 4   Arkansas  2,977,853   181         6.1
## 5 California 38,993,940 1,861         4.8
## 6   Colorado  5,448,819   176         3.2
```

```r
class(murders_raw$population)
```

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

```r
class(murders_raw$total)
```

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

Defining Strings: Single and Double Quotes and How to Escape

The textbook for this section is available here.

Key points

- Define a string by surrounding text with either single quotes or double quotes.
- To include a single quote inside a string, use double quotes on the outside. To include a double quote inside a string, use single quotes on the outside.
- The `cat()` function displays a string as it is represented inside R.
- To include a double quote inside of a string surrounded by double quotes, use the backslash () to escape the double quote. Escape a single quote to include it inside of a string defined by single quotes.
- We will see additional uses of the escape later.

Code

```r
s <- "Hello!"    # double quotes define a string
s <- 'Hello!'    # single quotes define a string
```

```r
s <- `Hello`    # backquotes do not
s <- "10""    # error - unclosed quotes
```

```r
s <- '10"'    # correct

# cat shows what the string actually looks like inside R
cat(s)
```

```
## 10"
```

```r
s <- "5'"
cat(s)
```

```
## 5'
```

```r
# to include both single and double quotes in string, escape with \
s <- '5'10"'    # error
s <- "5'10""    # error
```

```r
# to include both single and double quotes in string, escape with \
s <- '5\'10"'    # correct
cat(s)
```

```
## 5'10"
```

```
s <- "5'10\""    # correct
cat(s)
```

```
## 5'10"
```

stringr Package

The textbook for this section is available here.

Key points

- The main types of string processing tasks are detecting, locating, extracting and replacing elements of strings.
- The **stringr** package from the **tidyverse** includes a variety of string processing functions that begin with `str_` and take the string as the first argument, which makes them compatible with the pipe.

Code

```
# murders_raw defined in web scraping section

# direct conversion to numeric fails because of commas
murders_raw$population[1:3]
```

```
## [1] "4,853,875" "737,709"   "6,817,565"
```

```
as.numeric(murders_raw$population[1:3])
```

```
## Warning: NAs introduced by coercion
```

```
## [1] NA NA NA
```

Case Study 1: US Murders Data

The textbook for this section is available here.

Key points

- Use the `str_detect()` function to determine whether a string contains a certain pattern.
- Use the `str_replace_all()` function to replace all instances of one pattern with another pattern. To remove a pattern, replace with the empty string (`""`).
- The `parse_number()` function removes punctuation from strings and converts them to numeric.
- `mutate_at()` performs the same transformation on the specified column numbers.

Code

```
# murders_raw was defined in the web scraping section

# detect whether there are commas
commas <- function(x) any(str_detect(x, ","))
murders_raw %>% summarize_all(funs(commas))
```

```
## Warning: `funs()` is deprecated as of dplyr 0.8.0.
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
##   state population total murder_rate
## 1 FALSE       TRUE  TRUE       FALSE
```

```r
# replace commas with the empty string and convert to numeric
test_1 <- str_replace_all(murders_raw$population, ",", "")
test_1 <- as.numeric(test_1)

# parse_number also removes commas and converts to numeric
test_2 <- parse_number(murders_raw$population)
identical(test_1, test_2)
```

```
## [1] TRUE
```

```r
murders_new <- murders_raw %>% mutate_at(2:3, parse_number)
murders_new %>% head
```

```
##        state population total murder_rate
## 1    Alabama    4853875   348         7.2
## 2     Alaska     737709    59         8.0
## 3    Arizona    6817565   309         4.5
## 4   Arkansas    2977853   181         6.1
## 5 California   38993940  1861         4.8
## 6   Colorado    5448819   176         3.2
```

Assessment - String Processing Part 1

1. Which of the following is NOT an application of string parsing?

☐ A. Removing unwanted characters from text.
☐ B. Extracting numeric values from text.
☒ C. Formatting numbers and characters so they can easily be displayed in deliverables like papers and presentations.
☐ D. Splitting strings into multiple values.

2. Which of the following commands would not give you an error in R?

☒ A. `cat(" LeBron James is 6’8\" ")`

☐ B. `cat(' LeBron James is 6'8" ')`
☐ C. `` cat(` LeBron James is 6'8" `) ``
☐ D. `cat(" LeBron James is 6\'8" ")`

3. Which of the following are advantages of the **stringr** package over string processing functions in base R? Select all that apply.

☐ A. Base R functions are rarely used for string processing by data scientists so it's not worth learning them.
☒ B. Functions in stringr all start with "str_", which makes them easy to look up using autocomplete.
☒ C. Stringr functions work better with pipes.
☒ D. The order of arguments is more consistent in stringr functions than in base R.

4. You have a dataframe of monthly sales and profits in R

```
> head(dat)
# A tibble: 5 x 3
Month       Sales       Profit
<chr>       <chr>       <chr>
January     $128,568    $16,234
February    $109,523    $12,876
March       $115,468    $17,920
April       $122,274    $15,825
May         $117,921    $15,437
```

Which of the following commands could convert the sales and profits columns to numeric? Select all that apply.

☒ A. `dat %>% mutate_at(2:3, parse_number)`
☐ B. `dat %>% mutate_at(2:3, as.numeric)`
☐ C. `dat %>% mutate_all(parse_number)`
☒ D. `dat %>% mutate_at(2:3, funs(str_replace_all(., c("\\$|,"), ""))) %>%mutate_at(2:3, as.numeric)`

Case Study 2: Reported Heights

The textbook for this section is available here.

Key points

- In the raw heights data, many students did not report their height as the number of inches as requested. There are many entries with real height information but in the wrong format, which we can extract with string processing.
- When there are both text and numeric entries in a column, the column will be a character vector. Converting this column to numeric will result in NAs for some entries.
- To correct problematic entries, look for patterns that are shared across large numbers of entries, then define rules that identify those patterns and use these rules to write string processing tasks.
- Use `suppressWarnings()` to hide warning messages for a function.

Code

```r
# load raw heights data and inspect
data(reported_heights)
class(reported_heights$height)
```

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

```r
# convert to numeric, inspect, count NAs
x <- as.numeric(reported_heights$height)
```

```
## Warning: NAs introduced by coercion
```

```r
head(x)
```

```
## [1] 75 70 68 74 61 65
```

```r
sum(is.na(x))
```

```
## [1] 81
```

```r
# keep only entries that result in NAs
reported_heights %>% mutate(new_height = as.numeric(height)) %>%
  filter(is.na(new_height)) %>%
  head(n=10)
```

```
## Warning: Problem with `mutate()` input `new_height`.
## i NAs introduced by coercion
## i Input `new_height` is `as.numeric(height)`.
```

```
## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion
```

```
##             time_stamp    sex                 height new_height
## 1  2014-09-02 15:16:28   Male                  5' 4"         NA
## 2  2014-09-02 15:16:37 Female                  165cm         NA
## 3  2014-09-02 15:16:52   Male                    5'7         NA
## 4  2014-09-02 15:16:56   Male                  >9000         NA
## 5  2014-09-02 15:16:56   Male                   5'7"         NA
## 6  2014-09-02 15:17:09 Female                   5'3"         NA
## 7  2014-09-02 15:18:00   Male 5 feet and 8.11 inches         NA
## 8  2014-09-02 15:19:48   Male                   5'11         NA
## 9  2014-09-04 00:46:45   Male                  5'9''         NA
## 10 2014-09-04 10:29:44   Male                 5'10''         NA
```

```r
# calculate cutoffs that cover 99.999% of human population
alpha <- 1/10^6
qnorm(1-alpha/2, 69.1, 2.9)
```

```
## [1] 83.28575
```

```r
qnorm(alpha/2, 63.7, 2.7)
```

```
## [1] 50.49258
```

```r
# keep only entries that either result in NAs or are outside the plausible range of heights
not_inches <- function(x, smallest = 50, tallest = 84){
  inches <- suppressWarnings(as.numeric(x))
  ind <- is.na(inches) | inches < smallest | inches > tallest
  ind
}

# number of problematic entries
problems <- reported_heights %>%
  filter(not_inches(height)) %>%
  .$height
length(problems)
```

```
## [1] 292
```

```r
# 10 examples of x'y or x'y" or x'y\"
pattern <- "^\\d\\s*'\\s*\\d{1,2}\\.*\\d*'*\"*$"
str_subset(problems, pattern) %>% head(n=10) %>% cat
```

```
## 5' 4" 5'7 5'7" 5'3" 5'11 5'9'' 5'10'' 5' 10 5'5" 5'2"
```

```r
# 10 examples of x.y or x,y
pattern <- "^[4-6]\\s*[\\.|,]\\s*([0-9]|10|11)$"
str_subset(problems, pattern) %>% head(n=10) %>% cat
```

```
## 5.3 5.5 6.5 5.8 5.6 5,3 5.9 6,8 5.5 6.2
```

```r
# 10 examples of entries in cm rather than inches
ind <- which(between(suppressWarnings(as.numeric(problems))/2.54, 54, 81) )
ind <- ind[!is.na(ind)]
problems[ind] %>% head(n=10) %>% cat
```

```
## 150 175 177 178 163 175 178 165 165 180
```

Regex

The textbook for this section is available here through section 24.5.2.

Key points

- A regular expression (regex) is a way to describe a specific pattern of characters of text. A set of rules has been designed to do this specifically and efficiently.
- **stringr** functions can take a regex as a pattern.
- `str_detect()` indicates whether a pattern is present in a string.
- The main difference between a regex and a regular string is that a regex can include special characters.

- The | symbol inside a regex means "or".
- Use `'\\d'` to represent digits. The backlash is used to distinguish it from the character `'d'`. In R, you must use two backslashes for digits in regular expressions; in some other languages, you will only use one backslash for regex special characters.
- `str_view()` highlights the first occurrence of a pattern, and the `str_view_all()` function highlights all occurrences of the pattern.

Code

```
# detect whether a comma is present
pattern <- ","
str_detect(murders_raw$total, pattern)

# show the subset of strings including "cm"
str_subset(reported_heights$height, "cm")

# use the "or" symbol inside a regex (|)
yes <- c("180 cm", "70 inches")
no <- c("180", "70''")
s <- c(yes, no)
str_detect(s, "cm") | str_detect(s, "inches")
str_detect(s, "cm|inches")

# highlight the first occurrence of a pattern
str_view(s, pattern)

# highlight all instances of a pattern
str_view_all(s, pattern)
```

Character Classes, Anchors and Quantifiers

The textbook for this section is available here, here and here

Key points

- Define strings to test your regular expressions, including some elements that match and some that do not. This allows you to check for the two types of errors: failing to match and matching incorrectly.
- Square brackets define character classes: groups of characters that count as matching the pattern. You can use ranges to define character classes, such as `[0-9]` for digits and `[a-zA-Z]` for all letters.
- Anchors define patterns that must start or end at specific places. `^` and `$` represent the beginning and end of the string respectively.
- Curly braces are quantifiers that state how many times a certain character can be repeated in the pattern. `\\d{1,2}` matches exactly 1 or 2 consecutive digits.

Code

```
# s was defined in the previous section
yes <- c("5", "6", "5'10", "5 feet", "4'11")
no <- c("", ".", "Five", "six")
s <- c(yes, no)
pattern <- "\\d"
```

```r
# [56] means 5 or 6
str_view(s, "[56]")

# [4-7] means 4, 5, 6 or 7
yes <- as.character(4:7)
no <- as.character(1:3)
s <- c(yes, no)
str_detect(s, "[4-7]")

# ^ means start of string, $ means end of string
pattern <- "^\\d$"
yes <- c("1", "5", "9")
no <- c("12", "123", " 1", "a4", "b")
s <- c(yes, no)
str_view(s, pattern)

# curly braces define quantifiers: 1 or 2 digits
pattern <- "^\\d{1,2}$"
yes <- c("1", "5", "9", "12")
no <- c("123", "a4", "b")
str_view(c(yes, no), pattern)

# combining character class, anchors and quantifier
pattern <- "^[4-7]'\\d{1,2}\"$"
yes <- c("5'7\"", "6'2\"",  "5'12\"")
no <- c("6,2\"", "6.2\"","I am 5'11\"", "3'2\"", "64")
str_detect(yes, pattern)
str_detect(no, pattern)
```

Search and Replace with Regex

The textbook for this section is available:

- searching and replacing with regex.
- white space.
- quantifiers: *, +, ?.

Key points

- `str_replace()` replaces the first instance of the detected pattern with a specified string.
- Spaces are characters and R does not ignore them. Spaces are specified by the special character `\\s`.
- Additional quantifiers include `*`, `+` and `?`. `*` means 0 or more instances of the previous character. `?` means 0 or 1 instances. `+` means 1 or more instances.
- Before removing characters from strings with functions like `str_replace()` and `str_replace_all()`, consider whether that replacement would have unintended effects.

Code

```r
# number of entries matching our desired pattern
pattern <- "^[4-7]'\\d{1,2}\"$"
sum(str_detect(problems, pattern))
```

```r
# inspect examples of entries with problems
problems[c(2, 10, 11, 12, 15)] %>% str_view(pattern)
str_subset(problems, "inches")
str_subset(problems, "''")

# replace or remove feet/inches words before matching
pattern <- "^[4-7]'\\d{1,2}$"
problems %>%
  str_replace("feet|ft|foot", "'") %>% # replace feet, ft, foot with '
  str_replace("inches|in|''|\"", "") %>% # remove all inches symbols
  str_detect(pattern) %>%
  sum()

# R does not ignore whitespace
identical("Hi", "Hi ")

# \\s represents whitespace
pattern_2 <- "^[4-7]'\\s\\d{1,2}\"$"
str_subset(problems, pattern_2)

# * means 0 or more instances of a character
yes <- c("AB", "A1B", "A11B", "A111B", "A1111B")
no <- c("A2B", "A21B")
str_detect(yes, "A1*B")
str_detect(no, "A1*B")

# test how *, ? and + differ
data.frame(string = c("AB", "A1B", "A11B", "A111B", "A1111B"),
           none_or_more = str_detect(yes, "A1*B"),
           nore_or_once = str_detect(yes, "A1?B"),
           once_or_more = str_detect(yes, "A1+B"))

# update pattern by adding optional spaces before and after feet symbol
pattern <- "^[4-7]\\s*'\\s*\\d{1,2}$"
problems %>%
  str_replace("feet|ft|foot", "'") %>% # replace feet, ft, foot with '
  str_replace("inches|in|''|\"", "") %>% # remove all inches symbols
  str_detect(pattern) %>%
  sum()
```

Groups with Regex

The textbook for this section is available here.

Key Points

- Groups are defined using parentheses.
- Once we define groups, we can use the function `str_match()` to extract the values these groups define. `str_extract()` extracts only strings that match a pattern, not the values defined by groups.
- You can refer to the ith group with `\\i`. For example, refer to the value in the second group with `\\2`.

Code

```r
# define regex with and without groups
pattern_without_groups <- "^[4-7],\\d*$"
pattern_with_groups <-  "^([4-7]),(\\d*)$"

# create examples
yes <- c("5,9", "5,11", "6,", "6,1")
no <- c("5'9", ",", "2,8", "6.1.1")
s <- c(yes, no)

# demonstrate the effect of groups
str_detect(s, pattern_without_groups)
```

```
## [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

```r
str_detect(s, pattern_with_groups)
```

```
## [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

```r
# demonstrate difference between str_match and str_extract
str_match(s, pattern_with_groups)
```

```
##      [,1]   [,2] [,3]
## [1,] "5,9"  "5"  "9"
## [2,] "5,11" "5"  "11"
## [3,] "6,"   "6"  ""
## [4,] "6,1"  "6"  "1"
## [5,] NA     NA   NA
## [6,] NA     NA   NA
## [7,] NA     NA   NA
## [8,] NA     NA   NA
```

```r
str_extract(s, pattern_with_groups)
```

```
## [1] "5,9"  "5,11" "6,"   "6,1"  NA     NA     NA     NA
```

```r
# improve the pattern to recognize more events
pattern_with_groups <-  "^([4-7]),(\\d*)$"
yes <- c("5,9", "5,11", "6,", "6,1")
no <- c("5'9", ",", "2,8", "6.1.1")
s <- c(yes, no)
str_replace(s, pattern_with_groups, "\\1'\\2")
```

```
## [1] "5'9"   "5'11"  "6'"    "6'1"   "5'9"   ","     "2,8"   "6.1.1"
```

```r
# final pattern
pattern_with_groups <-"^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$"

# combine stringr commands with the pipe
str_subset(problems, pattern_with_groups) %>% head
```

```
## [1] "5.3"  "5.25" "5.5"  "6.5"  "5.8"  "5.6"
```

```
str_subset(problems, pattern_with_groups) %>%
  str_replace(pattern_with_groups, "\\1'\\2") %>% head
```

```
## [1] "5'3"  "5'25" "5'5"  "6'5"  "5'8"  "5'6"
```

Testing and Improving

The textbook for this section is available here.

Key points

- Wrangling with regular expressions is often an iterative process of testing the approach, looking for problematic entries, and improving the patterns.
- Use the pipe to connect **stringr** functions.
- It may not be worth writing code to correct every unique problem in the data, but string processing techniques are flexible enough for most needs.

Code

```
# function to detect entries with problems
not_inches_or_cm <- function(x, smallest = 50, tallest = 84){
    inches <- suppressWarnings(as.numeric(x))
    ind <- !is.na(inches) &
        ((inches >= smallest & inches <= tallest) |
             (inches/2.54 >= smallest & inches/2.54 <= tallest))
    !ind
}

# identify entries with problems
problems <- reported_heights %>%
  filter(not_inches_or_cm(height)) %>%
  .$height
length(problems)
```

```
## [1] 200
```

```
converted <- problems %>%
  str_replace("feet|foot|ft", "'") %>% #convert feet symbols to '
  str_replace("inches|in|''|\"", "") %>%  #remove inches symbols
  str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2") ##change format

# find proportion of entries that fit the pattern after reformatting
pattern <- "^[4-7]\\s*'\\s*\\d{1,2}$"
index <- str_detect(converted, pattern)
mean(index)
```

```
## [1] 0.615
```

```
converted[!index]     # show problems
```

```
##  [1] "6"             "165cm"         "511"           "6"
##  [5] "2"             ">9000"         "5 ' and 8.11 " "11111"
##  [9] "6"             "103.2"         "19"            "5"
## [13] "300"           "6'"            "6"             "Five ' eight "
## [17] "7"             "214"           "6"             "0.7"
## [21] "6"             "2'33"          "612"           "1,70"
## [25] "87"            "5'7.5"         "5'7.5"         "111"
## [29] "5' 7.78"       "12"            "6"             "yyy"
## [33] "89"            "34"            "25"            "6"
## [37] "6"             "22"            "684"           "6"
## [41] "1"             "1"             "6*12"          "87"
## [45] "6"             "1.6"           "120"           "120"
## [49] "23"            "1.7"           "6"             "5"
## [53] "69"            "5' 9 "         "5 ' 9 "        "6"
## [57] "6"             "86"            "708,661"       "5 ' 6 "
## [61] "6"             "649,606"       "10000"         "1"
## [65] "728,346"       "0"             "6"             "6"
## [69] "6"             "100"           "88"            "6"
## [73] "170 cm"        "7,283,465"     "5"             "5"
## [77] "34"
```

Assessment - String Processing Part 2

1. In the video, we use the function `not_inches` to identify heights that were incorrectly entered

```
not_inches <- function(x, smallest = 50, tallest = 84) {
  inches <- suppressWarnings(as.numeric(x))
  ind <- is.na(inches) | inches < smallest | inches > tallest
  ind
}
```

In this function, what TWO types of values are identified as not being correctly formatted in inches?

- ☐ A. Values that specifically contain apostrophes (‘), periods (.) or quotations (“).
- ☒ B. Values that result in NA’s when converted to numeric
- ☒ C. Values less than 50 inches or greater than 84 inches
- ☐ D. Values that are stored as a character class, because most are already classed as numeric.

2. Which of the following arguments, when passed to the function `not_inches`, would return the vector `c(FALSE)`?

- ☐ A. `c(175)`
- ☐ B. `c("5’8\"")`
- ☒ C. `c(70)`
- ☐ D. `c(85)` (the height of Shaquille O’Neal in inches)

3. Our function `not_inches` returns the object `ind`. Which answer correctly describes `ind`?

- ☒ A. `ind` is a logical vector of TRUE and FALSE, equal in length to the vector `x` (in the arguments list). TRUE indicates that a height entry is incorrectly formatted.
- ☐ B. `ind` is a logical vector of TRUE and FALSE, equal in length to the vector `x` (in the arguments list). TRUE indicates that a height entry is correctly formatted.
- ☐ C. `ind` is a data frame like our `reported_heights` table but with an extra column of TRUE or FALSE. TRUE indicates that a height entry is incorrectly formatted.
- ☐ D. `ind` is a numeric vector equal to `reported_heights$heights` but with incorrectly formatted heights replaced with NAs.

4. Given the following code

```
s <- c("70", "5 ft", "4'11", "", ".", "Six feet")
s
```

```
## [1] "70"       "5 ft"     "4'11"     ""         "."        "Six feet"
```

What `pattern` vector yields the following result?

```
str_view_all(s, pattern)
70
5 ft
4'11

.
Six feet
```

```
pattern <- "\\d|ft"
str_view_all(s, pattern)
```

- ☒ A. `pattern <- "\\d|ft"`
- ☐ B. `pattern <- "\d|ft"`
- ☐ C. `pattern <- "\\d\\d|ft"`
- ☐ D. `pattern <- "\\d|feet"`

5. You enter the following set of commands into your R console. What is your printed result?

```
animals <- c("cat", "puppy", "Moose", "MONKEY")
pattern <- "[a-z]"
str_detect(animals, pattern)
```

```
## [1]  TRUE  TRUE  TRUE FALSE
```

- ☐ A. TRUE
- ☐ B. TRUE TRUE TRUE TRUE
- ☒ C. TRUE TRUE TRUE FALSE
- ☐ D. TRUE TRUE FALSE FALSE

6. You enter the following set of commands into your R console. What is your printed result?

```r
animals <- c("cat", "puppy", "Moose", "MONKEY")
pattern <- "[A-Z]$"
str_detect(animals, pattern)
```

```
## [1] FALSE FALSE FALSE  TRUE
```

- ☐ A. FALSE FALSE FALSE FALSE
- ☐ B. FALSE FALSE TRUE TRUE
- ☒ C. FALSE FALSE FALSE TRUE
- ☐ D. TRUE TRUE TRUE FALSE

7. You enter the following set of commands into your R console. What is your printed result?

```r
animals <- c("cat", "puppy", "Moose", "MONKEY")
pattern <- "[a-z]{4,5}"
str_detect(animals, pattern)
```

```
## [1] FALSE  TRUE  TRUE FALSE
```

- ☒ A. FALSE TRUE TRUE FALSE
- ☐ B. TRUE TRUE FALSE FALSE
- ☐ C. FALSE FALSE FALSE TRUE
- ☐ D. TRUE TRUE TRUE FALSE

8. Given the following code

```r
animals <- c("moose", "monkey", "meerkat", "mountain lion")
------------
str_detect(animals, pattern)
```

Which TWO "pattern" vectors would yield the following result?

```
[1] TRUE TRUE TRUE TRUE
```

```r
animals <- c("moose", "monkey", "meerkat", "mountain lion")
pattern <- "mo*"
str_detect(animals, pattern)
```

```
## [1] TRUE TRUE TRUE TRUE
```

```r
animals <- c("moose", "monkey", "meerkat", "mountain lion")
pattern <- "mo?"
str_detect(animals, pattern)
```

```
## [1] TRUE TRUE TRUE TRUE
```

- ☒ A. `pattern <- "mo*"`
- ☒ B. `pattern <- "mo?"`
- ☐ C. `pattern <- "mo+"`

☐ D. pattern <- "moo*"

9. You are working on some data from different universities. You have the following vector

```
schools <- c("U. Kentucky", "Univ New Hampshire", "Univ. of Massachusetts", "University Georgia", "U Cal
schools
```

```
## [1] "U. Kentucky"                  "Univ New Hampshire"
## [3] "Univ. of Massachusetts"       "University Georgia"
## [5] "U California"                 "California State University"
```

You want to clean this data to match the full names of each university

```
> final
[1] "University of Kentucky"      "University of New Hampshire" "University of Massachusetts" "Universi
[5] "University of California"    "California State University"
```

What of the following commands could accomplish this?

```
schools %>%
  str_replace("^Univ\\.?\\s|^U\\.?\\s", "University ") %>%
  str_replace("^University of |^University ", "University of ")
```

```
## [1] "University of Kentucky"      "University of New Hampshire"
## [3] "University of Massachusetts" "University of Georgia"
## [5] "University of California"    "California State University"
```

☐ A.

```
schools %>%
  str_replace("Univ\\.?|U\\.?", "University ") %>%
  str_replace("^University of |^University ", "University of ")
```

☒ B.

```
schools %>%
  str_replace("^Univ\\.?\\s|^U\\.?\\s", "University ") %>%
  str_replace("^University of |^University ", "University of ")
```

☐ C.

```
schools %>%
  str_replace("^Univ\\.\\s|^U\\.\\s", "University") %>%
  str_replace("^University of |^University ", "University of ")
```

☐ D.

```
schools %>%
  str_replace("^Univ\\.?\\s|^U\\.?\\s", "University") %>%
  str_replace("University ", "University of ")
```

10. Rather than using the `pattern_with_groups` vector, you accidentally write in the following code

```
problems <- c("5.3", "5,5", "6 1", "5 .11", "5, 12")
pattern_with_groups <- "^([4-7])[,\\.](\\d*)$"
str_replace(problems, pattern_with_groups, "\\1'\\2")
```

```
## [1] "5'3"   "5'5"   "6 1"   "5 .11" "5, 12"
```

What is your result?

- ☒ A. [1] "5'3" "5'5" "6 1" "5 .11" "5, 12"
- ☐ B. [1] "5.3" "5,5" "6 1" "5 .11" "5, 12"
- ☐ C. [1] "5'3" "5'5" "6'1" "5 .11" "5, 12"
- ☐ D. [1] "5'3" "5'5" "6'1" "5'11" "5'12"

11. You notice your mistake and correct your pattern regex to the following

```
problems <- c("5.3", "5,5", "6 1", "5 .11", "5, 12")
pattern_with_groups <- "^([4-7])[,\\.\\s](\\d*)$"
str_replace(problems, pattern_with_groups, "\\1'\\2")
```

```
## [1] "5'3"   "5'5"   "6'1"   "5 .11" "5, 12"
```

What is your result?

- ☐ A. [1] "5'3" "5'5" "6 1" "5 .11" "5, 12"
- ☐ B. [1] "5.3" "5,5" "6 1" "5 .11" "5, 12"
- ☒ C. [1] "5'3" "5'5" "6'1" "5 .11" "5, 12"
- ☐ D. [1] "5'3" "5'5" "6'1" "5'11" "5'12"

12. In our example, we use the following code to detect height entries that do not match our pattern of x'y".

```
converted <- problems %>%
  str_replace("feet|foot|ft", "'") %>%
  str_replace("inches|in|''|\"", "") %>%
  str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2")

pattern <- "^[4-7]\\s*'\\s*\\d{1,2}$"
index <- str_detect(converted, pattern)
converted[!index]
```

```
## character(0)
```

Which answer best describes the differences between the regex string we use as an argument in `str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2")` and the regex string in `pattern <- "^[4-7]\\s*'\\s*\\d{1,2}$"`?

- ☐ A. The regex used in `str_replace()` looks for either a comma, period or space between the feet and inches digits, while the pattern regex just looks for an apostrophe; the regex in str_replace allows for one or more digits to be entered as inches, while the pattern regex only allows for one or two digits.
- ☐ B. The regex used in `str_replace()` allows for additional spaces between the feet and inches digits, but the pattern regex does not.
- ☐ C. The regex used in `str_replace()` looks for either a comma, period or space between the feet and inches digits, while the pattern regex just looks for an apostrophe; the regex in str_replace allows none or more digits to be entered as inches, while the pattern regex only allows for the number 1 or 2 to be used.
- ☒ D. The regex used in `str_replace()` looks for either a comma, period or space between the feet and inches digits, while the pattern regex just looks for an apostrophe; the regex in str_replace allows for none or more digits to be entered as inches, while the pattern regex only allows for one or two digits.

13. You notice a few entries that are not being properly converted using your `str_replace` and `str_detect` code

```
yes <- c("5 feet 7inches", "5 7")
no <- c("5ft 9 inches", "5 ft 9 inches")
s <- c(yes, no)

converted <- s %>%
  str_replace("feet|foot|ft", "'") %>%
  str_replace("inches|in|''|\"", "") %>%
  str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2")

pattern <- "^[4-7]\\s*'\\s*\\d{1,2}$"
str_detect(converted, pattern)
[1]  TRUE TRUE FALSE FALSE
```

It seems like the problem may be due to spaces around the words feet|foot|ft and inches|in. What is another way you could fix this problem?

```
yes <- c("5 feet 7inches", "5 7")
no <- c("5ft 9 inches", "5 ft 9 inches")
s <- c(yes, no)

converted <- s %>%
  str_replace("\\s*(feet|foot|ft)\\s*", "'") %>%
  str_replace("\\s*(inches|in|''|\")\\s*", "") %>%
  str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2")

pattern <- "^[4-7]\\s*'\\s*\\d{1,2}$"
str_detect(converted, pattern)
```

```
## [1] TRUE TRUE TRUE TRUE
```

- ☒ A.

```
converted <- s %>%
  str_replace("\\s*(feet|foot|ft)\\s*", "'") %>%
  str_replace("\\s*(inches|in|''|\")\\s*", "") %>%
  str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2")
```

☐ B.

```
converted <- s %>%
  str_replace("\\s+feet|foot|ft\\s+", "'") %>%
  str_replace("\\s+inches|in|''|\"\\s+", "") %>%
  str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2")
```

☐ C.

```
converted <- s %>%
  str_replace("\\s*|feet|foot|ft", "'") %>%
  str_replace("\\s*|inches|in|''|\"", "") %>%
  str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2")
```

☐ D.

```
converted <- s %>%
  str_replace_all("\\s", "") %>%
  str_replace("\\s|feet|foot|ft", "'") %>%
  str_replace("\\s|inches|in|''|\"", "") %>%
  str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2")
```

Separate with Regex

The textbook for this section is available here.

Key Point

- The `extract()` function behaves similarly to the separate() function but allows extraction of groups from regular expressions.

Code

```
# first example - normally formatted heights
s <- c("5'10", "6'1")
tab <- data.frame(x = s)

# the separate and extract functions behave similarly
tab %>% separate(x, c("feet", "inches"), sep = "'")
```

```
##   feet inches
## 1    5     10
## 2    6      1
```

```
tab %>% extract(x, c("feet", "inches"), regex = "(\\d)'(\\d{1,2})")
```

```
##   feet inches
## 1    5     10
## 2    6      1
```

```
# second example - some heights with unusual formats
s <- c("5'10", "6'1\"","5'8inches")
tab <- data.frame(x = s)

# separate fails because it leaves in extra characters, but extract keeps only the digits because of re
tab %>% separate(x, c("feet","inches"), sep = "'", fill = "right")
```

```
##   feet  inches
## 1    5      10
## 2    6      1"
## 3    5 8inches
```

```
tab %>% extract(x, c("feet", "inches"), regex = "(\\d)'(\\d{1,2})")
```

```
##   feet inches
## 1    5     10
## 2    6      1
## 3    5      8
```

Using Groups and Quantifiers

The textbook for this section is available here; through 24.10.

Four clear patterns of entries have arisen along with some other minor problems:

1. Many students measuring exactly 5 or 6 feet did not enter any inches. For example, **6'** - our pattern requires that inches be included.
2. Some students measuring exactly 5 or 6 feet entered just that number.
3. Some of the inches were entered with decimal points. For example **5'7.5''**. Our pattern only looks for two digits.
4. Some entires have spaces at the end, for example **5 ' 9**.
5. Some entries are in meters and some of these use European decimals: **1.6, 1,7**.
6. Two students added **cm**.
7. One student spelled out the numbers: **Five foot eight inches**. It is not necessarily clear that it is worth writing code to handle all these cases since they might be rare enough. However, some give us an opportunity to learn some more regex techniques so we will build a fix.

Case 1

For case 1, if we add a '0 to, for example, convert all 6 to 6'0, then our pattern will match. This can be done using groups using the following code:

```
yes <- c("5", "6", "5")
no <- c("5'", "5''", "5'4")
s <- c(yes, no)
str_replace(s, "^([4-7])$", "\\1'0")
```

The pattern says it has to start (^), be followed with a digit between 4 and 7, and then end there (**$**). The parenthesis defines the group that we pass as **\1** to the replace regex.

Cases 2 and 4

We can adapt this code slightly to handle case 2 as well which covers the entry **5'**. Note that the **5'** is left untouched by the code above. This is because the extra **'** makes the pattern not match since we have to end with a 5 or 6. To handle case 2, we want to permit the 5 or 6 to be followed by no or one symbol for feet. So we can simply add **'{0,1}** after the **'** to do this. We can also use the none or once special character **?**. As we saw previously, this is different from * which is none or more. We now see that this code also handles the fourth case as well:

```
str_replace(s, "^([56])'?$", "\\1'0")
```

Note that here we only permit 5 and 6 but not 4 and 7. This is because heights of exactly 5 and exactly 6 feet tall are quite common, so we assume those that typed 5 or 6 really meant either 60 or 72 inches. However, heights of exactly 4 or exactly 7 feet tall are so rare that, although we accept 84 as a valid entry, we assume that a 7 was entered in error.

Case 3

We can use quantifiers to deal with case 3. These entries are not matched because the inches include decimals and our pattern does not permit this. We need allow the second group to include decimals and not just digits. This means we must permit zero or one period **.** followed by zero or more digits. So we will use both **?** and *. Also remember that for this particular case, the period needs to be escaped since it is a special character (it means any character except a line break).

So we can adapt our pattern, currently `^[4-7]\\s*'\\s*\\d{1,2}$`, to permit a decimal at the end:

```
pattern <- "^[4-7]\\s*'\\s*(\\d+\\.?\\d*)$"
```

Case 5

Case 5, meters using commas, we can approach similarly to how we converted the x.y to x'y. A difference is that we require that the first digit is 1 or 2:

```
yes <- c("1,7", "1, 8", "2, " )
no <- c("5,8", "5,3,2", "1.7")
s <- c(yes, no)
str_replace(s, "^([12])\\s*,\\s*(\\d*)$", "\\1\\.\\2")
```

We will later check if the entries are meters using their numeric values.

Trimming

In general, spaces at the start or end of the string are uninformative. These can be particularly deceptive because sometimes they can be hard to see:

```
s <- "Hi "
cat(s)
identical(s, "Hi")
```

This is a general enough problem that there is a function dedicated to removing them: `str_trim`.

```r
str_trim("5 ' 9 ")
```

To upper and to lower case

One of the entries writes out numbers as words: **Five foot eight inches**. Although not efficient, we could add 12 extra **str_replace** to convert **zero** to **0**, **one** to **1**, and so on. To avoid having to write two separate operations for **Zero** and **zero**, **One** and **one**, etc., we can use the `str_to_lower()` function to make all words lower case first:

```r
s <- c("Five feet eight inches")
str_to_lower(s)
```

Putting it into a function

We are now ready to define a procedure that handles converting all the problematic cases.

We can now put all this together into a function that takes a string vector and tries to convert as many strings as possible to a single format. Below is a function that puts together the previous code replacements:

```r
convert_format <- function(s){
  s %>%
    str_replace("feet|foot|ft", "'") %>% #convert feet symbols to '
    str_replace_all("inches|in|''|\"|cm|and", "") %>%  #remove inches and other symbols
    str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2") %>% #change x.y, x,y x y
    str_replace("^([56])'?$", "\\1'0") %>% #add 0 when to 5 or 6
    str_replace("^([12])\\s*,\\s*(\\d*)$", "\\1\\.\\2") %>% #change european decimal
    str_trim() #remove extra space
}
```

We can also write a function that converts words to numbers:

```r
words_to_numbers <- function(s){
  str_to_lower(s) %>%
    str_replace_all("zero", "0") %>%
    str_replace_all("one", "1") %>%
    str_replace_all("two", "2") %>%
    str_replace_all("three", "3") %>%
    str_replace_all("four", "4") %>%
    str_replace_all("five", "5") %>%
    str_replace_all("six", "6") %>%
    str_replace_all("seven", "7") %>%
    str_replace_all("eight", "8") %>%
    str_replace_all("nine", "9") %>%
    str_replace_all("ten", "10") %>%
    str_replace_all("eleven", "11")
}
```

Now we can see which problematic entries remain:

```r
converted <- problems %>% words_to_numbers %>% convert_format
remaining_problems <- converted[not_inches_or_cm(converted)]
pattern <- "^[4-7]\\s*'\\s*\\d+\\.?\\d*$"
index <- str_detect(remaining_problems, pattern)
remaining_problems[!index]
```

Putting it All Together

We are now ready to put everything we've done so far together and wrangle our reported heights data as we try to recover as many heights as possible. The code is complex but we will break it down into parts.

We start by cleaning up the `height` column so that the heights are closer to a feet'inches format. We added an original heights column so we can compare before and after.

Let's start by writing a function that cleans up strings so that all the feet and inches formats use the same x'y format when appropriate.

```r
pattern <- "^([4-7])\\s*'\\s*(\\d+\\.?\\d*)$"

convert_format <- function(s){
  s %>%
    str_replace("feet|foot|ft", "'") %>% #convert feet symbols to '
    str_replace_all("inches|in|''|\"|cm|and", "") %>%  #remove inches and other symbols
    str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2") %>% #change x.y, x,y x y
    str_replace("^([56])'?$", "\\1'0") %>% #add 0 when to 5 or 6
    str_replace("^([12])\\s*,\\s*(\\d*)$", "\\1\\.\\2") %>% #change european decimal
    str_trim() #remove extra space
}

words_to_numbers <- function(s){
  str_to_lower(s) %>%
    str_replace_all("zero", "0") %>%
    str_replace_all("one", "1") %>%
    str_replace_all("two", "2") %>%
    str_replace_all("three", "3") %>%
    str_replace_all("four", "4") %>%
    str_replace_all("five", "5") %>%
    str_replace_all("six", "6") %>%
    str_replace_all("seven", "7") %>%
    str_replace_all("eight", "8") %>%
    str_replace_all("nine", "9") %>%
    str_replace_all("ten", "10") %>%
    str_replace_all("eleven", "11")
}

smallest <- 50
tallest <- 84
new_heights <- reported_heights %>%
  mutate(original = height,
         height = words_to_numbers(height) %>% convert_format()) %>%
  extract(height, c("feet", "inches"), regex = pattern, remove = FALSE) %>%
  mutate_at(c("height", "feet", "inches"), as.numeric) %>%
  mutate(guess = 12*feet + inches) %>%
  mutate(height = case_when(
    !is.na(height) & between(height, smallest, tallest) ~ height, #inches
    !is.na(height) & between(height/2.54, smallest, tallest) ~ height/2.54, #centimeters
    !is.na(height) & between(height*100/2.54, smallest, tallest) ~ height*100/2.54, #meters
    !is.na(guess) & inches < 12 & between(guess, smallest, tallest) ~ guess, #feet'inches
    TRUE ~ as.numeric(NA))) %>%
  select(-guess)
```

```
## Warning: Problem with `mutate()` input `height`.
## i NAs introduced by coercion
## i Input `height` is `.Primitive("as.double")(height)`.

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion
```

We can check all the entries we converted using the following code:

```
new_heights %>%
  filter(not_inches(original)) %>%
  select(original, height) %>%
  arrange(height) %>%
  View() # Open XQuartz-app to run this command
```

Let's take a look at the shortest students in our dataset using the following code:

```
new_heights %>% arrange(height) %>% head(n=7)
```

```
##            time_stamp    sex height feet inches original
## 1 2017-07-04 01:30:25   Male  50.00   NA     NA       50
## 2 2017-09-07 10:40:35   Male  50.00   NA     NA       50
## 3 2014-09-02 15:18:30 Female  51.00   NA     NA       51
## 4 2016-06-05 14:07:20 Female  52.00   NA     NA       52
## 5 2016-06-05 14:07:38 Female  52.00   NA     NA       52
## 6 2014-09-23 03:39:56 Female  53.00   NA     NA       53
## 7 2015-01-07 08:57:29   Male  53.77   NA     NA    53.77
```

We see heights of 53, 54, and 55. In the original heights column, we also have 51 and 52. These short heights are very rare and it is likely that the students actually meant 5'1, 5'2, 5'3, 5'4, and 5'5. But because we are not completely sure, we will leave them as reported.

String Splitting

The textbook for this section is available here.

Key Points

- The function `str_split()` splits a string into a character vector on a delimiter (such as a comma, space or underscore). By default, `str_split()` generates a list with one element for each original string. Use the function argument `simplify=TRUE` to have `str_split()` return a matrix instead.
- The `map()` function from the **purrr** package applies the same function to each element of a list. To extract the ith entry of each element x, use `map(x, i)`.
- `map()` always returns a list. Use `map_chr()` to return a character vector and `map_int()` to return an integer.

Code

```
# read raw murders data line by line
filename <- system.file("extdata/murders.csv", package = "dslabs")
lines <- readLines(filename)
lines %>% head()
```

```
## [1] "state,abb,region,population,total" "Alabama,AL,South,4779736,135"
## [3] "Alaska,AK,West,710231,19"          "Arizona,AZ,West,6392017,232"
## [5] "Arkansas,AR,South,2915918,93"      "California,CA,West,37253956,1257"
```

```
# split at commas with str_split function, remove row of column names
x <- str_split(lines, ",")
x %>% head()
```

```
## [[1]]
## [1] "state"      "abb"        "region"     "population" "total"
##
## [[2]]
## [1] "Alabama" "AL"      "South"   "4779736" "135"
##
## [[3]]
## [1] "Alaska" "AK"     "West"   "710231" "19"
##
## [[4]]
## [1] "Arizona" "AZ"      "West"    "6392017" "232"
##
## [[5]]
## [1] "Arkansas" "AR"       "South"    "2915918"  "93"
##
## [[6]]
## [1] "California" "CA"         "West"       "37253956"   "1257"
```

```
col_names <- x[[1]]
x <- x[-1]

# extract first element of each list entry
if(!require(purrr)) install.packages("purrr")

library(purrr)
map(x, function(y) y[1]) %>% head()
```

```
## [[1]]
## [1] "Alabama"
##
## [[2]]
## [1] "Alaska"
##
## [[3]]
## [1] "Arizona"
##
## [[4]]
## [1] "Arkansas"
##
## [[5]]
## [1] "California"
##
## [[6]]
## [1] "Colorado"
```

```
map(x, 1) %>% head()
```

```
## [[1]]
## [1] "Alabama"
##
## [[2]]
## [1] "Alaska"
##
## [[3]]
## [1] "Arizona"
##
## [[4]]
## [1] "Arkansas"
##
## [[5]]
## [1] "California"
##
## [[6]]
## [1] "Colorado"
```

```
# extract columns 1-5 as characters, then convert to proper format - NOTE: DIFFERENT FROM VIDEO
dat <- data.frame(parse_guess(map_chr(x, 1)),
                  parse_guess(map_chr(x, 2)),
                  parse_guess(map_chr(x, 3)),
                  parse_guess(map_chr(x, 4)),
                  parse_guess(map_chr(x, 5))) %>%
  setNames(col_names)

dat %>% head
```

```
##        state abb region population total
## 1    Alabama  AL  South    4779736   135
## 2     Alaska  AK   West     710231    19
## 3    Arizona  AZ   West    6392017   232
## 4   Arkansas  AR  South    2915918    93
## 5 California  CA   West   37253956  1257
## 6   Colorado  CO   West    5029196    65
```

```
# more efficient code for the same thing
dat <- x %>%
  transpose() %>%
  map( ~ parse_guess(unlist(.))) %>%
  setNames(col_names) %>%
  as.data.frame()

# the simplify argument makes str_split return a matrix instead of a list
x <- str_split(lines, ",", simplify = TRUE)
col_names <- x[1,]
x <- x[-1,]
x %>% as_data_frame() %>%
  setNames(col_names) %>%
  mutate_all(parse_guess)
```

```
## Warning: `as_data_frame()` is deprecated as of tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.

## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if `.name_repair` is
## Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.

## # A tibble: 51 x 5
##    state                abb   region    population total
##    <chr>                <chr> <chr>          <dbl> <dbl>
##  1 Alabama              AL    South        4779736   135
##  2 Alaska               AK    West          710231    19
##  3 Arizona              AZ    West         6392017   232
##  4 Arkansas             AR    South        2915918    93
##  5 California           CA    West        37253956  1257
##  6 Colorado             CO    West         5029196    65
##  7 Connecticut          CT    Northeast    3574097    97
##  8 Delaware             DE    South         897934    38
##  9 District of Columbia DC    South         601723    99
## 10 Florida              FL    South       19687653   669
## # ... with 41 more rows
```

Case Study - Extracting a Table from a PDF

The textbook for this section is available here.

One of the datasets provided in `dslabs` shows scientific funding rates by gender in the Netherlands:

```
data("research_funding_rates")
research_funding_rates
```

```
##           discipline applications_total applications_men applications_women
## 1   Chemical sciences                122               83                 39
## 2   Physical sciences                174              135                 39
## 3             Physics                 76               67                  9
## 4          Humanities                396              230                166
## 5  Technical sciences                251              189                 62
## 6   Interdisciplinary                183              105                 78
## 7 Earth/life sciences                282              156                126
## 8     Social sciences                834              425                409
## 9    Medical sciences                505              245                260
##   awards_total awards_men awards_women success_rates_total success_rates_men
## 1           32         22           10                26.2              26.5
## 2           35         26            9                20.1              19.3
## 3           20         18            2                26.3              26.9
## 4           65         33           32                16.4              14.3
## 5           43         30           13                17.1              15.9
## 6           29         12           17                15.8              11.4
## 7           56         38           18                19.9              24.4
```

```
## 8         112         65           47                 13.4               15.3
## 9          75         46           29                 14.9               18.8
##   success_rates_women
## 1                25.6
## 2                23.1
## 3                22.2
## 4                19.3
## 5                21.0
## 6                21.8
## 7                14.3
## 8                11.5
## 9                11.2
```

The data come from a paper published in the prestigious journal PNAS. However, the data are not provided in a spreadsheet; they are in a table in a PDF document. We could extract the numbers by hand, but this could lead to human error. Instead we can try to wrangle the data using R.

**Downloading the data*

We start by downloading the PDF document then importing it into R using the following code:

```
if(!require(pdftools)) install.packages("pdftools")
```

```
## Loading required package: pdftools
```

```
## Using poppler version 0.73.0
```

```
library("pdftools")
temp_file <- tempfile()
url <- "http://www.pnas.org/content/suppl/2015/09/16/1510159112.DCSupplemental/pnas.201510159SI.pdf"
download.file(url, temp_file)
txt <- pdf_text(temp_file)
file.remove(temp_file)
```

```
## [1] TRUE
```

If we examine the object `txt` we notice that it is a character vector with an entry for each page. So we keep the page we want using the following code:

```
raw_data_research_funding_rates <- txt[2]
```

The steps above can actually be skipped because we include the raw data in the `dslabs` package as well:

```
data("raw_data_research_funding_rates")
```

Looking at the download

Examining this object,

```
raw_data_research_funding_rates %>% head
```

```
## [1] "                          Table S1. Numbers of applications and awarded grants, along with success
```

we see that it is a long string. Each line on the page, including the table rows, is separated by the symbol for newline: `\n`.

We can therefore can create a list with the lines of the text as elements:

```
tab <- str_split(raw_data_research_funding_rates, "\n")
```

Because we start off with just one element in the string, we end up with a list with just one entry:

```
tab <- tab[[1]]
```

By examining this object,

```
tab %>% head
```

```
## [1] "                            Table S1. Numbers of applications and awarded grants, along with success
## [2] "                            female applicants, by scientific discipline"
## [3] "                                                               Applications, n                  Awards
## [4] "                            Discipline              Total      Men       Women          Total    Men
## [5] "                           Total                    2,823     1,635      1,188           467      290
## [6] "                            Chemical sciences         122        83         39            32       22
```

we see that the information for the column names is the third and fourth entires:

```
the_names_1 <- tab[3]
the_names_2 <- tab[4]
```

In the table, the column information is spread across two lines. We want to create one vector with one name for each column. We can do this using some of the functions we have just learned.

Extracting the table data

Let's start with the first line:

```
the_names_1
```

```
## [1] "                                                               Applications, n                  Awards
```

We want to remove the leading space and everything following the comma. We can use regex for the latter. Then we can obtain the elements by splitting using the space. We want to split only when there are 2 or more spaces to avoid splitting `success rate`. So we use the regex `\\s{2,}` as follows:

```
the_names_1 <- the_names_1 %>%
  str_trim() %>%
  str_replace_all(",\\s.", "") %>%
  str_split("\\s{2,}", simplify = TRUE)
the_names_1
```

```
##      [,1]           [,2]     [,3]
## [1,] "Applications" "Awards" "Success rates"
```

Now let's look at the second line:

```
the_names_2
```

```
## [1] "                          Discipline              Total      Men       Women           Total     Men
```

Here we want to trim the leading space and then split by space as we did for the first line:

```
the_names_2 <- the_names_2 %>%
  str_trim() %>%
  str_split("\\s+", simplify = TRUE)
the_names_2
```

```
##      [,1]         [,2]    [,3]  [,4]    [,5]    [,6]  [,7]    [,8]    [,9]
## [1,] "Discipline" "Total" "Men" "Women" "Total" "Men" "Women" "Total" "Men"
##      [,10]
## [1,] "Women"
```

Now we can join these to generate one name for each column:

```
tmp_names <- str_c(rep(the_names_1, each = 3), the_names_2[-1], sep = "_")
the_names <- c(the_names_2[1], tmp_names) %>%
  str_to_lower() %>%
  str_replace_all("\\s", "_")
the_names
```

```
##  [1] "discipline"          "applications_total"  "applications_men"
##  [4] "applications_women"  "awards_total"        "awards_men"
##  [7] "awards_women"        "success_rates_total" "success_rates_men"
## [10] "success_rates_women"
```

Now we are ready to get the actual data. By examining the `tab` object, we notice that the information is in lines 6 through 14. We can use `str_split()` again to achieve our goal:

```
new_research_funding_rates <- tab[6:14] %>%
  str_trim %>%
  str_split("\\s{2,}", simplify = TRUE) %>%
  data.frame(stringsAsFactors = FALSE) %>%
  setNames(the_names) %>%
  mutate_at(-1, parse_number)
new_research_funding_rates %>% head()
```

```
##            discipline applications_total applications_men applications_women
## 1   Chemical sciences                122               83                 39
## 2   Physical sciences                174              135                 39
## 3             Physics                 76               67                  9
## 4          Humanities                396              230                166
## 5  Technical sciences                251              189                 62
## 6   Interdisciplinary                183              105                 78
##   awards_total awards_men awards_women success_rates_total success_rates_men
## 1           32         22           10                26.2              26.5
## 2           35         26            9                20.1              19.3
```

```
## 3          20        18           2                 26.3              26.9
## 4          65        33          32                 16.4              14.3
## 5          43        30          13                 17.1              15.9
## 6          29        12          17                 15.8              11.4
##   success_rates_women
## 1                25.6
## 2                23.1
## 3                22.2
## 4                19.3
## 5                21.0
## 6                21.8
```

We can see that the objects are identical:

```r
identical(research_funding_rates, new_research_funding_rates)
```

```
## [1] TRUE
```

Recoding

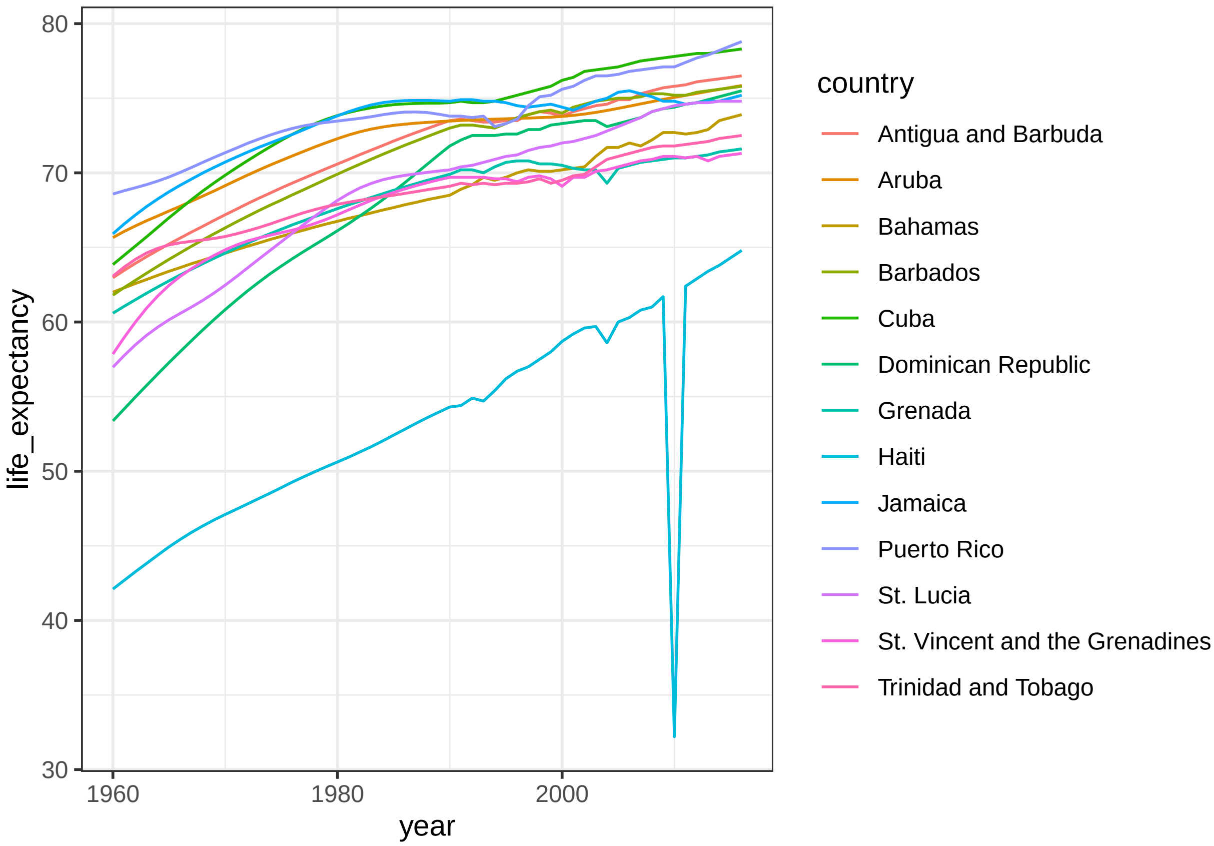
The textbook for this section is available here.

Key points

- Change long factor names with the `recode()` function from the **tidyverse**.
- Other similar functions include `recode_factor()` and `fct_recoder()` in the **forcats** package in the **tidyverse**. The same result could be obtained using the `case_when()` function, but `recode()` is more efficient to write.

Code

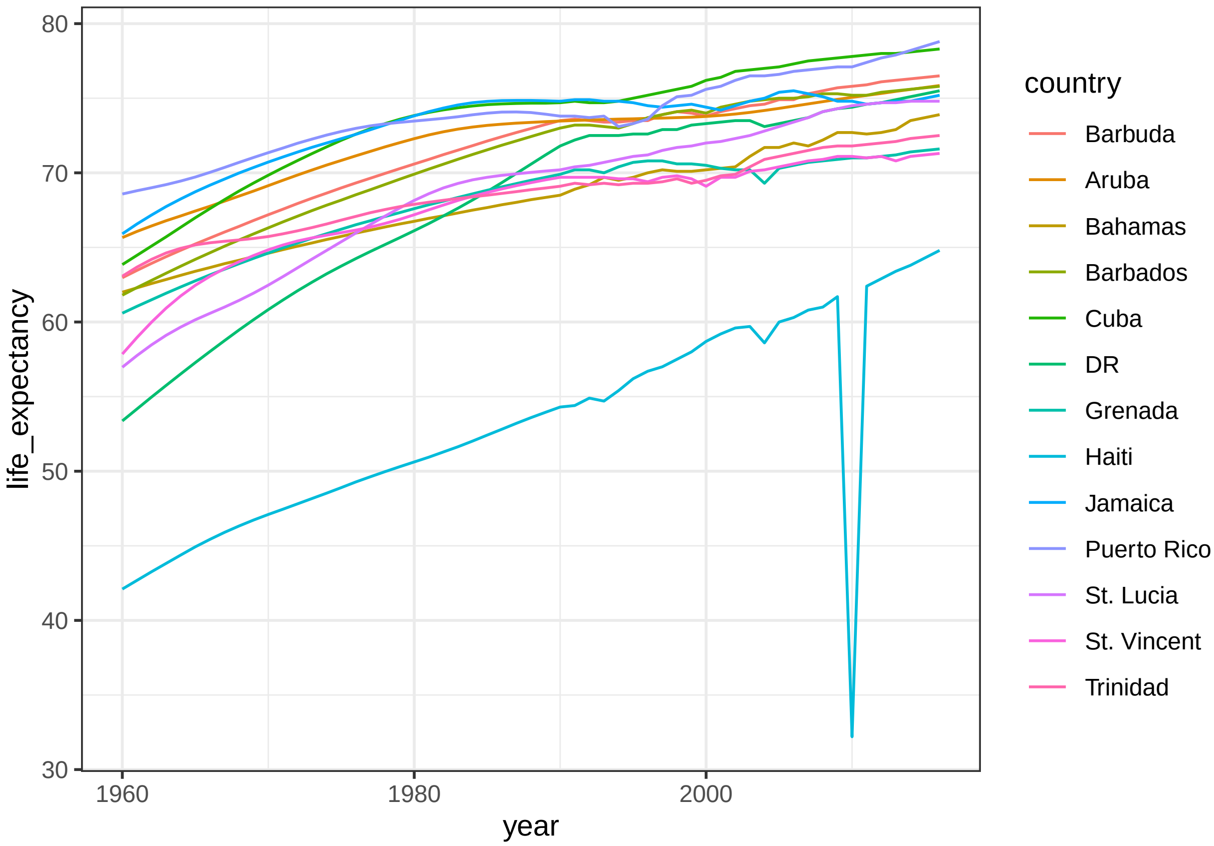
```r
# life expectancy time series for Caribbean countries
gapminder %>%
  filter(region=="Caribbean") %>%
  ggplot(aes(year, life_expectancy, color = country)) +
  geom_line()
```




```
# display long country names
gapminder %>%
  filter(region=="Caribbean") %>%
  filter(str_length(country) >= 12) %>%
  distinct(country)
```

```
##                           country
## 1             Antigua and Barbuda
## 2              Dominican Republic
## 3 St. Vincent and the Grenadines
## 4             Trinidad and Tobago
```

```
# recode long country names and remake plot
gapminder %>% filter(region=="Caribbean") %>%
  mutate(country = recode(country,
                          'Antigua and Barbuda'="Barbuda",
                          'Dominican Republic' = "DR",
                          'St. Vincent and the Grenadines' = "St. Vincent",
                          'Trinidad and Tobago' = "Trinidad")) %>%
  ggplot(aes(year, life_expectancy, color = country)) +
  geom_line()
```



Assessment - String Processing Part 3

Want even more practice with regular expressions? Complete the lessons and exercises in the RegexOne online interactive tutorial!

1.

```
s <- c("5'10", "6'1\"", "5'8inches", "5'7.5")
tab <- data.frame(x = s)
```

If you use the extract code from our video, the decimal point is dropped. What modification of the code would allow you to put the decimals in a third column called "decimal"?

```
extract(data = tab, col = x, into = c("feet", "inches", "decimal"),
regex = "(\\d)'(\\d{1,2})(\\.\\d+)?")
```

```
##   feet inches decimal
## 1    5     10
## 2    6      1
## 3    5      8
## 4    5      7      .5
```

☐ A.

```
extract(data = tab, col = x, into = c("feet", "inches", "decimal"), regex = "(\\d)'(\\d{1,2})(\\.)?"
```

☐ B.

```
extract(data = tab, col = x, into = c("feet", "inches", "decimal"), regex = "(\\d)'(\\d{1,2})(\\.\\d+)"
```

☐ C.

```
extract(data = tab, col = x, into = c("feet", "inches", "decimal"), regex = "(\\d)'(\\d{1,2})\\.\\d+?"
```

☒ D.

```
extract(data = tab, col = x, into = c("feet", "inches", "decimal"), regex = "(\\d)'(\\d{1,2})(\\.\\d+)?
```

2. You have the following table, `schedule`

```
>schedule
day         staff
Monday      Mandy, Chris and Laura
Tuesday     Steve, Ruth and Frank
```

You want to turn this into a more useful data frame.

Which two commands would properly split the text in the "staff" column into each individual name? Select ALL that apply.

☐ A. `str_split(schedule$staff, ",|and")`
☒ B. `str_split(schedule$staff, ", | and ")`
☒ C. `str_split(schedule$staff, ",\\s|\\sand\\s")`
☐ D. `str_split(schedule$staff, "\\s?(,|and)\\s?")`

3. You have the following table, `schedule`

```
> schedule
day         staff
Monday      Mandy, Chris and Laura
Tuesday     Steve, Ruth and Frank
```

What code would successfully turn your "Schedule" table into the following tidy table

```
< tidy
day      staff
<chr>    <chr>
Monday   Mandy
Monday   Chris
Monday   Laura
Tuesday  Steve
Tuesday  Ruth
Tuesday  Frank
```

☒ A.

```
tidy <- schedule %>%
  mutate(staff = str_split(staff, ", | and ")) %>%
  unnest()
```

☐ B.

```
tidy <- separate(schedule, staff, into = c("s1","s2","s3"), sep = ",") %>%
  gather(key = s, value = staff, s1:s3)
```

☐ C.

```
tidy <- schedule %>%
  mutate(staff = str_split(staff, ", | and ", simplify = TRUE)) %>%
  unnest()
```

4. Using the gapminder data, you want to recode countries longer than 12 letters in the region "Middle Africa" to their abbreviations in a new column, "country_short". Which code would accomplish this?

```
dat <- gapminder %>% filter(region == "Middle Africa") %>%
  mutate(country_short = recode(country,
                        "Central African Republic" = "CAR",
                        "Congo, Dem. Rep." = "DRC",
                        "Equatorial Guinea" = "Eq. Guinea"))
dat
```

```
##                     country year infant_mortality life_expectancy fertility
## 1                    Angola 1960            208.0           35.98      7.32
## 2                  Cameroon 1960            166.9           43.46      5.65
## 3  Central African Republic 1960            165.5           37.43      5.84
## 4                      Chad 1960               NA           40.95      6.25
## 5          Congo, Dem. Rep. 1960            174.0           43.90      6.00
## 6               Congo, Rep. 1960            110.6           48.25      5.88
## 7         Equatorial Guinea 1960               NA           37.69      5.51
## 8                     Gabon 1960               NA           38.83      4.38
## 9                    Angola 1961               NA           36.53      7.35
## 10                 Cameroon 1961            163.3           44.00      5.71
## 11 Central African Republic 1961            162.9           37.89      5.87
## 12                     Chad 1961               NA           41.35      6.27
## 13         Congo, Dem. Rep. 1961               NA           44.25      6.02
## 14              Congo, Rep. 1961            108.4           48.88      5.92
## 15        Equatorial Guinea 1961               NA           38.04      5.52
## 16                    Gabon 1961               NA           39.15      4.46
## 17                   Angola 1962               NA           37.08      7.39
## 18                 Cameroon 1962            160.5           44.53      5.77
## 19 Central African Republic 1962            160.4           38.36      5.89
## 20                     Chad 1962               NA           41.76      6.29
## 21         Congo, Dem. Rep. 1962               NA           44.61      6.03
## 22              Congo, Rep. 1962            106.1           49.47      5.97
## 23        Equatorial Guinea 1962               NA           38.38      5.53
## 24                    Gabon 1962               NA           39.56      4.54
## 25                   Angola 1963               NA           37.63      7.41
```

```
## 26                 Cameroon 1963          158.1          45.07      5.83
## 27 Central African Republic 1963          157.6          38.85      5.91
## 28                     Chad 1963             NA          42.17      6.30
## 29         Congo, Dem. Rep. 1963             NA          44.98      6.05
## 30              Congo, Rep. 1963          104.2          50.04      6.01
## 31        Equatorial Guinea 1963             NA          38.73      5.55
## 32                    Gabon 1963             NA          40.07      4.62
## 33                   Angola 1964             NA          38.18      7.43
## 34                 Cameroon 1964          155.5          45.59      5.89
## 35 Central African Republic 1964          154.7          39.36      5.93
## 36                     Chad 1964             NA          42.58      6.32
## 37         Congo, Dem. Rep. 1964             NA          45.36      6.07
## 38              Congo, Rep. 1964          102.1          50.55      6.06
## 39        Equatorial Guinea 1964             NA          39.08      5.57
## 40                    Gabon 1964             NA          40.70      4.69
## 41                   Angola 1965             NA          38.74      7.43
## 42                 Cameroon 1965          152.2          46.13      5.95
## 43 Central African Republic 1965          151.7          39.92      5.94
## 44                     Chad 1965             NA          43.01      6.34
## 45         Congo, Dem. Rep. 1965             NA          45.77      6.09
## 46              Congo, Rep. 1965           99.7          51.02      6.10
## 47        Equatorial Guinea 1965             NA          39.44      5.60
## 48                    Gabon 1965             NA          41.42      4.77
## 49                   Angola 1966             NA          39.28      7.42
## 50                 Cameroon 1966          148.1          46.67      6.01
## 51 Central African Republic 1966          148.6          40.50      5.95
## 52                     Chad 1966             NA          43.48      6.36
## 53         Congo, Dem. Rep. 1966             NA          46.20      6.11
## 54              Congo, Rep. 1966           97.3          51.45      6.14
## 55        Equatorial Guinea 1966             NA          39.78      5.62
## 56                    Gabon 1966             NA          42.21      4.83
## 57                   Angola 1967             NA          39.84      7.40
## 58                 Cameroon 1967          143.1          47.22      6.06
## 59 Central African Republic 1967          145.6          41.15      5.95
## 60                     Chad 1967             NA          43.98      6.39
## 61         Congo, Dem. Rep. 1967             NA          46.66      6.14
## 62              Congo, Rep. 1967           94.9          51.84      6.17
## 63        Equatorial Guinea 1967             NA          40.13      5.64
## 64                    Gabon 1967             NA          43.06      4.90
## 65                   Angola 1968             NA          40.39      7.38
## 66                 Cameroon 1968          137.3          47.79      6.11
## 67 Central African Republic 1968          142.6          41.84      5.95
## 68                     Chad 1968             NA          44.54      6.43
## 69         Congo, Dem. Rep. 1968             NA          47.14      6.16
## 70              Congo, Rep. 1968           92.7          52.21      6.21
## 71        Equatorial Guinea 1968             NA          40.48      5.66
## 72                    Gabon 1968             NA          43.90      4.96
## 73                   Angola 1969             NA          40.95      7.34
## 74                 Cameroon 1969          131.5          48.37      6.16
## 75 Central African Republic 1969          139.8          42.57      5.95
## 76                     Chad 1969          137.3          45.12      6.48
## 77         Congo, Dem. Rep. 1969          150.7          47.63      6.19
## 78              Congo, Rep. 1969           90.6          52.54      6.24
## 79        Equatorial Guinea 1969             NA          40.82      5.67
```

```
## 80                    Gabon 1969            NA          44.74      5.02
## 81                   Angola 1970         180.0          41.50      7.30
## 82                 Cameroon 1970         126.2          48.97      6.21
## 83  Central African Republic 1970         137.0          43.36      5.95
## 84                     Chad 1970         135.9          45.72      6.53
## 85         Congo, Dem. Rep. 1970         149.0          48.13      6.21
## 86              Congo, Rep. 1970          88.5          52.85      6.26
## 87        Equatorial Guinea 1970            NA          41.17      5.68
## 88                    Gabon 1970            NA          45.55      5.08
## 89                   Angola 1971            NA          42.06      7.26
## 90                 Cameroon 1971         121.6          49.59      6.25
## 91  Central African Republic 1971         134.5          44.19      5.95
## 92                     Chad 1971         134.5          46.33      6.58
## 93         Congo, Dem. Rep. 1971         147.5          48.60      6.24
## 94              Congo, Rep. 1971          86.5          53.14      6.28
## 95        Equatorial Guinea 1971            NA          41.52      5.68
## 96                    Gabon 1971            NA          46.35      5.14
## 97                   Angola 1972            NA          42.62      7.23
## 98                 Cameroon 1972         118.2          50.22      6.29
## 99  Central African Republic 1972         132.1          45.04      5.95
## 100                    Chad 1972         131.8          46.91      6.64
## 101        Congo, Dem. Rep. 1972         145.9          49.05      6.27
## 102             Congo, Rep. 1972          84.4          53.42      6.30
## 103       Equatorial Guinea 1972            NA          41.87      5.68
## 104                   Gabon 1972            NA          47.13      5.21
## 105                  Angola 1973            NA          43.17      7.21
## 106                Cameroon 1973         116.0          50.85      6.32
## 107 Central African Republic 1973         129.9          45.91      5.95
## 108                    Chad 1973         131.2          47.47      6.69
## 109        Congo, Dem. Rep. 1973         144.2          49.46      6.30
## 110             Congo, Rep. 1973          82.5          53.69      6.31
## 111       Equatorial Guinea 1973            NA          42.21      5.68
## 112                   Gabon 1973            NA          47.90      5.28
## 113                  Angola 1974            NA          43.71      7.19
## 114                Cameroon 1974         114.7          51.49      6.36
## 115 Central African Republic 1974         127.9          46.77      5.95
## 116                    Chad 1974         130.6          47.98      6.74
## 117        Congo, Dem. Rep. 1974         142.5          49.83      6.33
## 118             Congo, Rep. 1974          80.9          53.94      6.32
## 119       Equatorial Guinea 1974            NA          42.56      5.68
## 120                   Gabon 1974            NA          48.68      5.34
## 121                  Angola 1975            NA          44.22      7.19
## 122                Cameroon 1975         114.3          52.13      6.39
## 123 Central African Republic 1975         126.0          47.60      5.95
## 124                    Chad 1975         130.0          48.45      6.78
## 125        Congo, Dem. Rep. 1975         140.7          50.17      6.37
## 126             Congo, Rep. 1975          79.2          54.20      6.33
## 127       Equatorial Guinea 1975            NA          42.91      5.67
## 128                   Gabon 1975            NA          49.45      5.41
## 129                  Angola 1976            NA          44.68      7.19
## 130                Cameroon 1976         114.2          52.74      6.43
## 131 Central African Republic 1976         124.2          48.36      5.95
## 132                    Chad 1976         129.4          48.89      6.82
## 133        Congo, Dem. Rep. 1976         138.9          50.49      6.40
```

```
## 134            Congo, Rep. 1976            77.6            54.45      6.32
## 135      Equatorial Guinea 1976              NA            43.28      5.68
## 136                  Gabon 1976              NA            50.23      5.48
## 137                 Angola 1977              NA            45.12      7.19
## 138               Cameroon 1977           113.9            53.36      6.47
## 139 Central African Republic 1977         122.6            49.07      5.95
## 140                   Chad 1977           128.8            49.31      6.86
## 141       Congo, Dem. Rep. 1977           137.1            50.80      6.44
## 142            Congo, Rep. 1977            76.0            54.71      6.30
## 143      Equatorial Guinea 1977              NA            43.65      5.68
## 144                  Gabon 1977              NA            51.01      5.54
## 145                 Angola 1978              NA            45.50      7.19
## 146               Cameroon 1978           113.0            53.95      6.52
## 147 Central African Republic 1978         121.0            49.70      5.95
## 148                   Chad 1978           128.1            49.72      6.89
## 149       Congo, Dem. Rep. 1978           135.4            51.11      6.49
## 150            Congo, Rep. 1978            74.4            54.97      6.27
## 151      Equatorial Guinea 1978              NA            44.04      5.69
## 152                  Gabon 1978            79.0            51.81      5.60
## 153                 Angola 1979              NA            45.84      7.20
## 154               Cameroon 1979           111.4            54.52      6.56
## 155 Central African Republic 1979         119.6            50.21      5.95
## 156                   Chad 1979           127.4            50.14      6.93
## 157       Congo, Dem. Rep. 1979           133.7            51.43      6.54
## 158            Congo, Rep. 1979            72.8            55.22      6.23
## 159      Equatorial Guinea 1979              NA            44.44      5.71
## 160                  Gabon 1979            76.6            52.61      5.65
## 161                 Angola 1980           138.3            46.14      7.20
## 162               Cameroon 1980           109.2            55.06      6.61
## 163 Central African Republic 1980         118.4            50.61      5.95
## 164                   Chad 1980           126.6            50.56      6.96
## 165       Congo, Dem. Rep. 1980           132.0            51.76      6.59
## 166            Congo, Rep. 1980            71.2            55.45      6.18
## 167      Equatorial Guinea 1980              NA            44.85      5.73
## 168                  Gabon 1980            74.2            53.42      5.68
## 169                 Angola 1981           137.5            46.42      7.20
## 170               Cameroon 1981           106.3            55.56      6.65
## 171 Central African Republic 1981         117.4            50.86      5.96
## 172                   Chad 1981           125.7            50.97      6.99
## 173       Congo, Dem. Rep. 1981           130.5            52.09      6.64
## 174            Congo, Rep. 1981            69.6            55.65      6.11
## 175      Equatorial Guinea 1981              NA            45.26      5.75
## 176                  Gabon 1981            71.9            54.24      5.71
## 177                 Angola 1982           136.8            46.69      7.20
## 178               Cameroon 1982           103.1            56.03      6.68
## 179 Central African Republic 1982         116.7            50.96      5.96
## 180                   Chad 1982           124.8            51.38      7.02
## 181       Congo, Dem. Rep. 1982           129.0            52.41      6.69
## 182            Congo, Rep. 1982            68.0            55.81      6.04
## 183      Equatorial Guinea 1982           145.1            45.69      5.78
## 184                  Gabon 1982            69.8            55.07      5.72
## 185                 Angola 1983           136.0            46.96      7.21
## 186               Cameroon 1983            99.5            56.45      6.70
## 187 Central African Republic 1983         116.2            50.95      5.96
```

```
## 188                     Chad 1983            123.8           51.78      7.06
## 189          Congo, Dem. Rep. 1983            127.7           52.72      6.74
## 190               Congo, Rep. 1983             66.3           55.93      5.95
## 191        Equatorial Guinea 1983            145.4           46.11      5.80
## 192                    Gabon 1983             67.9           55.88      5.72
## 193                   Angola 1984            135.3           47.23      7.21
## 194                 Cameroon 1984             95.8           56.83      6.71
## 195 Central African Republic 1984            115.9           50.81      5.95
## 196                     Chad 1984            122.7           52.15      7.09
## 197          Congo, Dem. Rep. 1984            126.3           53.00      6.79
## 198               Congo, Rep. 1984             64.6           55.98      5.86
## 199        Equatorial Guinea 1984            142.7           46.52      5.83
## 200                    Gabon 1984             66.2           56.66      5.71
## 201                   Angola 1985            134.9           47.50      7.21
## 202                 Cameroon 1985             92.4           57.17      6.70
## 203 Central African Republic 1985            115.7           50.57      5.94
## 204                     Chad 1985            121.5           52.51      7.12
## 205          Congo, Dem. Rep. 1985            125.2           53.28      6.85
## 206               Congo, Rep. 1985             63.1           55.94      5.77
## 207        Equatorial Guinea 1985            140.1           46.92      5.85
## 208                    Gabon 1985             64.7           57.40      5.69
## 209                   Angola 1986            134.4           47.75      7.21
## 210                 Cameroon 1986             89.4           57.48      6.67
## 211 Central African Republic 1986            115.6           50.21      5.93
## 212                     Chad 1986            120.4           52.81      7.16
## 213          Congo, Dem. Rep. 1986            124.0           53.55      6.90
## 214               Congo, Rep. 1986             61.8           55.79      5.68
## 215        Equatorial Guinea 1986            137.7           47.33      5.87
## 216                    Gabon 1986             63.5           58.04      5.65
## 217                   Angola 1987            134.1           47.99      7.20
## 218                 Cameroon 1987             87.2           57.75      6.63
## 219 Central African Republic 1987            115.5           49.80      5.90
## 220                     Chad 1987            119.3           53.09      7.20
## 221          Congo, Dem. Rep. 1987            122.9           53.81      6.96
## 222               Congo, Rep. 1987             60.9           55.54      5.59
## 223        Equatorial Guinea 1987            135.3           47.73      5.88
## 224                    Gabon 1987             62.5           58.58      5.61
## 225                   Angola 1988            133.8           48.20      7.19
## 226                 Cameroon 1988             85.8           58.01      6.57
## 227 Central African Republic 1988            115.4           49.34      5.87
## 228                     Chad 1988            118.1           53.33      7.24
## 229          Congo, Dem. Rep. 1988            121.8           54.07      7.02
## 230               Congo, Rep. 1988             60.4           55.21      5.50
## 231        Equatorial Guinea 1988            132.9           48.14      5.89
## 232                    Gabon 1988             61.7           59.00      5.55
## 233                   Angola 1989            133.6           48.40      7.18
## 234                 Cameroon 1989             85.2           58.22      6.51
## 235 Central African Republic 1989            115.4           48.86      5.83
## 236                     Chad 1989            117.0           53.52      7.28
## 237          Congo, Dem. Rep. 1989            120.8           54.31      7.08
## 238               Congo, Rep. 1989             60.4           54.78      5.42
## 239        Equatorial Guinea 1989            130.4           48.52      5.90
## 240                    Gabon 1989             61.1           59.32      5.49
## 241                   Angola 1990            133.5           48.60      7.17
```

```
## 242                 Cameroon 1990           85.6          58.40     6.43
## 243 Central African Republic 1990          115.3          48.40     5.78
## 244                     Chad 1990          115.8          53.70     7.31
## 245         Congo, Dem. Rep. 1990          119.8          54.50     7.13
## 246              Congo, Rep. 1990           60.9          54.30     5.35
## 247        Equatorial Guinea 1990          127.9          48.90     5.90
## 248                    Gabon 1990           60.5          59.50     5.42
## 249                   Angola 1991          133.5          49.30     7.14
## 250                 Cameroon 1991           86.7          58.20     6.35
## 251 Central African Republic 1991          115.2          48.10     5.73
## 252                     Chad 1991          114.7          54.30     7.35
## 253         Congo, Dem. Rep. 1991          118.7          54.40     7.18
## 254              Congo, Rep. 1991           61.8          54.40     5.29
## 255        Equatorial Guinea 1991          125.5          48.70     5.90
## 256                    Gabon 1991           60.0          59.80     5.34
## 257                   Angola 1992          133.5          49.60     7.12
## 258                 Cameroon 1992           88.2          57.90     6.26
## 259 Central African Republic 1992          115.1          48.00     5.69
## 260                     Chad 1992          113.7          53.90     7.38
## 261         Congo, Dem. Rep. 1992          117.7          54.30     7.22
## 262              Congo, Rep. 1992           63.0          54.40     5.24
## 263        Equatorial Guinea 1992          123.1          48.70     5.90
## 264                    Gabon 1992           59.6          60.20     5.26
## 265                   Angola 1993          133.4          48.40     7.09
## 266                 Cameroon 1993           89.9          57.40     6.17
## 267 Central African Republic 1993          115.2          47.50     5.65
## 268                     Chad 1993          112.6          54.00     7.40
## 269         Congo, Dem. Rep. 1993          116.8          54.30     7.25
## 270              Congo, Rep. 1993           64.6          53.50     5.20
## 271        Equatorial Guinea 1993          120.8          48.60     5.90
## 272                    Gabon 1993           59.1          60.10     5.17
## 273                   Angola 1994          133.2          50.00     7.05
## 274                 Cameroon 1994           91.5          57.00     6.08
## 275 Central African Republic 1994          115.4          47.20     5.62
## 276                     Chad 1994          111.7          53.60     7.42
## 277         Congo, Dem. Rep. 1994          115.8          54.30     7.27
## 278              Congo, Rep. 1994           66.6          53.20     5.17
## 279        Equatorial Guinea 1994          118.5          48.50     5.90
## 280                    Gabon 1994           58.6          59.90     5.08
## 281                   Angola 1995          132.8          50.90     7.02
## 282                 Cameroon 1995           93.0          56.50     5.99
## 283 Central African Republic 1995          115.4          46.70     5.60
## 284                     Chad 1995          110.9          53.60     7.43
## 285         Congo, Dem. Rep. 1995          114.9          54.00     7.27
## 286              Congo, Rep. 1995           68.8          52.60     5.15
## 287        Equatorial Guinea 1995          116.5          48.50     5.90
## 288                    Gabon 1995           58.2          59.80     4.99
## 289                   Angola 1996          132.3          51.30     6.98
## 290                 Cameroon 1996           94.1          56.20     5.90
## 291 Central African Republic 1996          115.4          46.30     5.57
## 292                     Chad 1996          110.0          53.00     7.43
## 293         Congo, Dem. Rep. 1996          113.8          51.80     7.25
## 294              Congo, Rep. 1996           71.2          52.20     5.15
## 295        Equatorial Guinea 1996          114.3          48.90     5.89
```

```
## 296                    Gabon 1996          57.8          59.60      4.90
## 297                   Angola 1997         131.5          51.70      6.95
## 298                 Cameroon 1997          94.7          55.50      5.82
## 299 Central African Republic 1997         115.1          45.90      5.55
## 300                     Chad 1997         109.1          52.50      7.42
## 301         Congo, Dem. Rep. 1997         112.5          53.20      7.23
## 302              Congo, Rep. 1997          73.5          46.30      5.14
## 303        Equatorial Guinea 1997         112.1          50.30      5.87
## 304                    Gabon 1997          57.3          59.90      4.81
## 305                   Angola 1998         130.6          51.80      6.91
## 306                 Cameroon 1998          94.7          55.00      5.75
## 307 Central African Republic 1998         114.7          45.70      5.52
## 308                     Chad 1998         108.0          52.10      7.41
## 309         Congo, Dem. Rep. 1998         110.9          53.50      7.19
## 310              Congo, Rep. 1998          75.3          49.90      5.14
## 311        Equatorial Guinea 1998         109.7          51.20      5.85
## 312                    Gabon 1998          56.8          60.00      4.74
## 313                   Angola 1999         129.5          51.80      6.88
## 314                 Cameroon 1999          93.8          54.70      5.68
## 315 Central African Republic 1999         114.2          45.50      5.49
## 316                     Chad 1999         106.9          51.70      7.38
## 317         Congo, Dem. Rep. 1999         109.3          54.00      7.14
## 318              Congo, Rep. 1999          76.5          51.60      5.14
## 319        Equatorial Guinea 1999         107.3          52.00      5.81
## 320                    Gabon 1999          56.3          59.70      4.66
## 321                   Angola 2000         128.3          52.30      6.84
## 322                 Cameroon 2000          91.9          54.30      5.62
## 323 Central African Republic 2000         113.6          45.30      5.45
## 324                     Chad 2000         105.7          51.50      7.35
## 325         Congo, Dem. Rep. 2000         107.4          54.30      7.09
## 326              Congo, Rep. 2000          76.6          52.50      5.13
## 327        Equatorial Guinea 2000         104.8          52.90      5.77
## 328                    Gabon 2000          55.6          59.30      4.60
## 329                   Angola 2001         126.9          52.50      6.81
## 330                 Cameroon 2001          89.3          54.20      5.57
## 331 Central African Republic 2001         112.9          45.20      5.39
## 332                     Chad 2001         104.6          51.70      7.32
## 333         Congo, Dem. Rep. 2001         105.3          54.50      7.03
## 334              Congo, Rep. 2001          75.5          53.50      5.13
## 335        Equatorial Guinea 2001         102.3          54.00      5.73
## 336                    Gabon 2001          54.7          59.00      4.54
## 337                   Angola 2002         125.5          53.30      6.78
## 338                 Cameroon 2002          86.2          54.20      5.52
## 339 Central African Republic 2002         112.1          45.20      5.33
## 340                     Chad 2002         103.4          51.90      7.27
## 341         Congo, Dem. Rep. 2002         103.1          54.70      6.96
## 342              Congo, Rep. 2002          73.3          54.30      5.13
## 343        Equatorial Guinea 2002          99.7          54.90      5.68
## 344                    Gabon 2002          53.7          59.40      4.49
## 345                   Angola 2003         124.1          53.90      6.74
## 346                 Cameroon 2003          83.1          54.30      5.46
## 347 Central African Republic 2003         111.3          45.20      5.26
## 348                     Chad 2003         102.3          52.10      7.21
## 349         Congo, Dem. Rep. 2003         100.9          54.90      6.89
```

```
## 350             Congo, Rep. 2003              70.0             55.00      5.13
## 351       Equatorial Guinea 2003              97.1             55.30      5.63
## 352                   Gabon 2003              52.6             59.40      4.45
## 353                  Angola 2004             122.8             54.50      6.70
## 354                Cameroon 2004              80.3             54.40      5.41
## 355 Central African Republic 2004             110.4             45.40      5.18
## 356                    Chad 2004             101.3             52.60      7.15
## 357        Congo, Dem. Rep. 2004              98.7             55.90      6.81
## 358             Congo, Rep. 2004              66.1             55.80      5.12
## 359       Equatorial Guinea 2004              94.4             55.90      5.57
## 360                   Gabon 2004              51.4             59.40      4.41
## 361                  Angola 2005             121.2             55.20      6.66
## 362                Cameroon 2005              77.8             54.90      5.36
## 363 Central African Republic 2005             109.3             45.50      5.09
## 364                    Chad 2005             100.4             53.00      7.07
## 365        Congo, Dem. Rep. 2005              96.3             56.40      6.73
## 366             Congo, Rep. 2005              61.8             56.70      5.12
## 367       Equatorial Guinea 2005              91.7             56.00      5.52
## 368                   Gabon 2005              50.2             60.10      4.37
## 369                  Angola 2006             119.4             55.70      6.60
## 370                Cameroon 2006              75.3             55.40      5.30
## 371 Central African Republic 2006             108.1             45.80      5.00
## 372                    Chad 2006              99.4             53.10      6.99
## 373        Congo, Dem. Rep. 2006              93.9             56.80      6.64
## 374             Congo, Rep. 2006              57.4             57.80      5.11
## 375       Equatorial Guinea 2006              89.0             56.80      5.46
## 376                   Gabon 2006              49.0             60.90      4.34
## 377                  Angola 2007             117.1             56.20      6.52
## 378                Cameroon 2007              73.1             55.70      5.24
## 379 Central African Republic 2007             106.9             46.20      4.90
## 380                    Chad 2007              98.1             54.00      6.90
## 381        Congo, Dem. Rep. 2007              91.5             57.10      6.55
## 382             Congo, Rep. 2007              53.1             58.30      5.11
## 383       Equatorial Guinea 2007              86.3             57.10      5.39
## 384                   Gabon 2007              47.4             61.60      4.30
## 385                  Angola 2008             114.7             56.70      6.43
## 386                Cameroon 2008              70.8             56.60      5.17
## 387 Central African Republic 2008             105.5             46.80      4.81
## 388                    Chad 2008              96.7             54.30      6.81
## 389        Congo, Dem. Rep. 2008              89.2             57.50      6.45
## 390             Congo, Rep. 2008              49.0             58.80      5.10
## 391       Equatorial Guinea 2008              83.7             57.50      5.31
## 392                   Gabon 2008              45.7             61.70      4.28
## 393                  Angola 2009             112.2             57.10      6.33
## 394                Cameroon 2009              68.8             57.30      5.09
## 395 Central African Republic 2009             103.6             47.60      4.72
## 396                    Chad 2009              95.1             55.20      6.70
## 397        Congo, Dem. Rep. 2009              86.9             57.90      6.35
## 398             Congo, Rep. 2009              45.3             59.80      5.09
## 399       Equatorial Guinea 2009              81.2             58.00      5.23
## 400                   Gabon 2009              44.2             62.10      4.25
## 401                  Angola 2010             109.6             57.60      6.22
## 402                Cameroon 2010              66.2             57.80      5.02
## 403 Central African Republic 2010             101.7             47.90      4.63
```

```
## 404                     Chad 2010            93.6           55.80      6.60
## 405         Congo, Dem. Rep. 2010            84.8           58.40      6.25
## 406              Congo, Rep. 2010            42.2           60.40      5.07
## 407        Equatorial Guinea 2010            78.9           58.60      5.14
## 408                    Gabon 2010            42.8           63.00      4.21
## 409                   Angola 2011           106.8           58.10      6.10
## 410                 Cameroon 2011            64.4           58.10      4.94
## 411 Central African Republic 2011            99.7           48.10      4.54
## 412                     Chad 2011            91.9           56.10      6.49
## 413         Congo, Dem. Rep. 2011            82.6           58.80      6.15
## 414              Congo, Rep. 2011            39.6           60.90      5.05
## 415        Equatorial Guinea 2011            76.6           58.70      5.04
## 416                    Gabon 2011            41.3           63.30      4.18
## 417                   Angola 2012           104.1           58.50      5.98
## 418                 Cameroon 2012            62.4           58.50      4.86
## 419 Central African Republic 2012            97.7           48.50      4.45
## 420                     Chad 2012            90.2           56.30      6.38
## 421         Congo, Dem. Rep. 2012            80.5           59.10      6.04
## 422              Congo, Rep. 2012            37.6           61.30      5.01
## 423        Equatorial Guinea 2012            74.3           59.40      4.95
## 424                    Gabon 2012            39.7           63.90      4.14
## 425                   Angola 2013           101.4           58.80      5.86
## 426                 Cameroon 2013            60.4           59.00      4.78
## 427 Central African Republic 2013            96.1           47.80      4.37
## 428                     Chad 2013            88.4           56.60      6.26
## 429         Congo, Dem. Rep. 2013            78.3           59.60      5.93
## 430              Congo, Rep. 2013            35.9           61.50      4.97
## 431        Equatorial Guinea 2013            72.2           60.50      4.85
## 432                    Gabon 2013            38.0           64.40      4.09
## 433                   Angola 2014            98.8           59.20      5.75
## 434                 Cameroon 2014            58.6           59.10      4.70
## 435 Central African Republic 2014            93.5           48.20      4.28
## 436                     Chad 2014            86.7           56.80      6.15
## 437         Congo, Dem. Rep. 2014            76.5           60.10      5.83
## 438              Congo, Rep. 2014            34.4           61.50      4.92
## 439        Equatorial Guinea 2014            70.3           61.00      4.75
## 440                    Gabon 2014            37.0           65.00      4.03
## 441                   Angola 2015            96.0           59.60      5.65
## 442                 Cameroon 2015            57.1           59.40      4.63
## 443 Central African Republic 2015            91.5           49.60      4.20
## 444                     Chad 2015            85.0           57.40      6.04
## 445         Congo, Dem. Rep. 2015            74.5           60.80      5.72
## 446              Congo, Rep. 2015            33.2           61.50      4.86
## 447        Equatorial Guinea 2015            68.2           61.00      4.65
## 448                    Gabon 2015            36.1           65.90      3.97
## 449                   Angola 2016              NA           60.00        NA
## 450                 Cameroon 2016              NA           59.70        NA
## 451 Central African Republic 2016              NA           51.04        NA
## 452                     Chad 2016              NA           58.01        NA
## 453         Congo, Dem. Rep. 2016              NA           61.51        NA
## 454              Congo, Rep. 2016              NA           61.50        NA
## 455        Equatorial Guinea 2016              NA           61.00        NA
## 456                    Gabon 2016              NA           66.81        NA
##     population         gdp continent        region country_short
```

```
## 1      5270844          NA    Africa Middle Africa       Angola
## 2      5361367  2537944080    Africa Middle Africa     Cameroon
## 3      1503501   534982718    Africa Middle Africa          CAR
## 4      3002596   750173439    Africa Middle Africa         Chad
## 5     15248246  4992962083    Africa Middle Africa          DRC
## 6      1013581   626127041    Africa Middle Africa  Congo, Rep.
## 7       252115          NA    Africa Middle Africa   Eq. Guinea
## 8       499189   887289809    Africa Middle Africa        Gabon
## 9      5367287          NA    Africa Middle Africa       Angola
## 10     5474509  2785779139    Africa Middle Africa     Cameroon
## 11     1529229   561479896    Africa Middle Africa          CAR
## 12     3061423   760658941    Africa Middle Africa         Chad
## 13    15637715  4451156989    Africa Middle Africa          DRC
## 14     1039966   678538008    Africa Middle Africa  Congo, Rep.
## 15      255100          NA    Africa Middle Africa   Eq. Guinea
## 16      504174  1018309175    Africa Middle Africa        Gabon
## 17     5465905          NA    Africa Middle Africa       Angola
## 18     5593768  2870510257    Africa Middle Africa     Cameroon
## 19     1556656   540628554    Africa Middle Africa          CAR
## 20     3122357   801431143    Africa Middle Africa         Chad
## 21    16041247  5394833319    Africa Middle Africa          DRC
## 22     1067611   713837650    Africa Middle Africa  Congo, Rep.
## 23      257940          NA    Africa Middle Africa   Eq. Guinea
## 24      509806  1094165180    Africa Middle Africa        Gabon
## 25     5565808          NA    Africa Middle Africa       Angola
## 26     5719135  2977940547    Africa Middle Africa     Cameroon
## 27     1585765   536804991    Africa Middle Africa          CAR
## 28     3184775   788612621    Africa Middle Africa         Chad
## 29    16461914  5676119396    Africa Middle Africa          DRC
## 30     1096502   685074987    Africa Middle Africa  Congo, Rep.
## 31      260990          NA    Africa Middle Africa   Eq. Guinea
## 32      516270  1160826485    Africa Middle Africa        Gabon
## 33     5665701          NA    Africa Middle Africa       Angola
## 34     5850454  3083572854    Africa Middle Africa     Cameroon
## 35     1616515   547972474    Africa Middle Africa          CAR
## 36     3247798   768811034    Africa Middle Africa         Chad
## 37    16903899  5537609393    Africa Middle Africa          DRC
## 38     1126602   711222903    Africa Middle Africa  Congo, Rep.
## 39      264743          NA    Africa Middle Africa   Eq. Guinea
## 40      523793  1213695790    Africa Middle Africa        Gabon
## 41     5765025          NA    Africa Middle Africa       Angola
## 42     5987671  3146047697    Africa Middle Africa     Cameroon
## 43     1648830   553164891    Africa Middle Africa          CAR
## 44     3310921   773471780    Africa Middle Africa         Chad
## 45    17369859  5592838673    Africa Middle Africa          DRC
## 46     1157905   737370784    Africa Middle Africa  Congo, Rep.
## 47      269427          NA    Africa Middle Africa   Eq. Guinea
## 48      532512  1314837134    Africa Middle Africa        Gabon
## 49     5863568          NA    Africa Middle Africa       Angola
## 50     6130990  3291236402    Africa Middle Africa     Cameroon
## 51     1682874   556732800    Africa Middle Africa          CAR
## 52     3373563   759494431    Africa Middle Africa         Chad
## 53    17861860  5971780635    Africa Middle Africa          DRC
## 54     1190361   747391524    Africa Middle Africa  Congo, Rep.
```

```
## 55     275470          NA    Africa Middle Africa    Eq. Guinea
## 56     542562  1374110052    Africa Middle Africa         Gabon
## 57    5962831          NA    Africa Middle Africa        Angola
## 58    6280743  2932094527    Africa Middle Africa      Cameroon
## 59    1718558   582768491    Africa Middle Africa           CAR
## 60    3436227   765321282    Africa Middle Africa          Chad
## 61   18378189  5912914485    Africa Middle Africa           DRC
## 62    1224041   763208362    Africa Middle Africa   Congo, Rep.
## 63     282445          NA    Africa Middle Africa    Eq. Guinea
## 64     553829  1430656781    Africa Middle Africa         Gabon
## 65    6066094          NA    Africa Middle Africa        Angola
## 66    6437157  3118174741    Africa Middle Africa      Cameroon
## 67    1755260   590951707    Africa Middle Africa           CAR
## 68    3500778   761822089    Africa Middle Africa          Chad
## 69   18913177  6169103240    Africa Middle Africa           DRC
## 70    1259190   821451282    Africa Middle Africa   Congo, Rep.
## 71     288701          NA    Africa Middle Africa    Eq. Guinea
## 72     565878  1466549111    Africa Middle Africa         Gabon
## 73    6177703          NA    Africa Middle Africa        Angola
## 74    6600479  3271011438    Africa Middle Africa      Cameroon
## 75    1792150   632858495    Africa Middle Africa           CAR
## 76    3569778   814245430    Africa Middle Africa          Chad
## 77   19458874  6744608852    Africa Middle Africa           DRC
## 78    1296137   883458739    Africa Middle Africa   Congo, Rep.
## 79     292014          NA    Africa Middle Africa    Eq. Guinea
## 80     578114  1585088962    Africa Middle Africa         Gabon
## 81    6300969          NA    Africa Middle Africa        Angola
## 82    6770967  3372153343    Africa Middle Africa      Cameroon
## 83    1828710   647622869    Africa Middle Africa           CAR
## 84    3644911   829387598    Africa Middle Africa          Chad
## 85   20009902  6728080745    Africa Middle Africa           DRC
## 86    1335090   939633199    Africa Middle Africa   Congo, Rep.
## 87     290905          NA    Africa Middle Africa    Eq. Guinea
## 88     590119  1722664256    Africa Middle Africa         Gabon
## 89    6437645          NA    Africa Middle Africa        Angola
## 90    6948847  3489494427    Africa Middle Africa      Cameroon
## 91    1864757   654941830    Africa Middle Africa           CAR
## 92    3727382   810746064    Africa Middle Africa          Chad
## 93   20563111  7132103209    Africa Middle Africa           DRC
## 94    1376189  1012483298    Africa Middle Africa   Congo, Rep.
## 95     284915          NA    Africa Middle Africa    Eq. Guinea
## 96     601734  1899387747    Africa Middle Africa         Gabon
## 97    6587647          NA    Africa Middle Africa        Angola
## 98    7134374  3582798039    Africa Middle Africa      Cameroon
## 99    1900702   654936703    Africa Middle Africa           CAR
## 100   3816299   820066531    Africa Middle Africa          Chad
## 101  21120996  7142882350    Africa Middle Africa           DRC
## 102   1419305  1099734561    Africa Middle Africa   Congo, Rep.
## 103    274906          NA    Africa Middle Africa    Eq. Guinea
## 104    613129  2114720779    Africa Middle Africa         Gabon
## 105   6750215          NA    Africa Middle Africa        Angola
## 106   7327874  3774681265    Africa Middle Africa      Cameroon
## 107   1937383   667306943    Africa Middle Africa           CAR
## 108   3908729   751337714    Africa Middle Africa          Chad
```

```
## 109   21690604  7724118393    Africa Middle Africa          DRC
## 110    1464052  1190255791    Africa Middle Africa   Congo, Rep.
## 111     262399          NA    Africa Middle Africa    Eq. Guinea
## 112     624625  2330050819    Africa Middle Africa         Gabon
## 113    6923749          NA    Africa Middle Africa        Angola
## 114    7529704  4179865452    Africa Middle Africa      Cameroon
## 115    1975968   709608469    Africa Middle Africa           CAR
## 116    4000511   788617984    Africa Middle Africa          Chad
## 117   22282079  7965928553    Africa Middle Africa           DRC
## 118    1509880  1284113659    Africa Middle Africa   Congo, Rep.
## 119     249587          NA    Africa Middle Africa    Eq. Guinea
## 120     636702  3250120203    Africa Middle Africa         Gabon
## 121    7107334          NA    Africa Middle Africa        Angola
## 122    7740196  4649893779    Africa Middle Africa      Cameroon
## 123    2017379   712482410    Africa Middle Africa           CAR
## 124    4088858   859674573    Africa Middle Africa          Chad
## 125   22902275  7569095386    Africa Middle Africa           DRC
## 126    1556406  1383396150    Africa Middle Africa   Congo, Rep.
## 127     238240          NA    Africa Middle Africa    Eq. Guinea
## 128     649719  3873822005    Africa Middle Africa         Gabon
## 129    7299508          NA    Africa Middle Africa        Angola
## 130    7959500  4394375680    Africa Middle Africa      Cameroon
## 131    2061552   751187608    Africa Middle Africa           CAR
## 132    4173070   885299454    Africa Middle Africa          Chad
## 133   23556784  7167251955    Africa Middle Africa           DRC
## 134    1603446  1396072025    Africa Middle Africa   Congo, Rep.
## 135     228491          NA    Africa Middle Africa    Eq. Guinea
## 136     663774  5253884186    Africa Middle Africa         Gabon
## 137    7501320          NA    Africa Middle Africa        Angola
## 138    8187840  4998157253    Africa Middle Africa      Cameroon
## 139    2108417   779779920    Africa Middle Africa           CAR
## 140    4254770   905082575    Africa Middle Africa          Chad
## 141   24242643  7221779949    Africa Middle Africa           DRC
## 142    1651134  1271075722    Africa Middle Africa   Congo, Rep.
## 143     220352          NA    Africa Middle Africa    Eq. Guinea
## 144     678786  4592835688    Africa Middle Africa         Gabon
## 145    7717139          NA    Africa Middle Africa        Angola
## 146    8425707  6097902039    Africa Middle Africa      Cameroon
## 147    2158844   789209202    Africa Middle Africa           CAR
## 148    4336389   900831998    Africa Middle Africa          Chad
## 149   24948113  6835754757    Africa Middle Africa           DRC
## 150    1699781  1351912922    Africa Middle Africa   Congo, Rep.
## 151     215284          NA    Africa Middle Africa    Eq. Guinea
## 152     694734  3488295169    Africa Middle Africa         Gabon
## 153    7952882          NA    Africa Middle Africa        Angola
## 154    8673666  6465917670    Africa Middle Africa      Cameroon
## 155    2213888   769754633    Africa Middle Africa           CAR
## 156    4421448   707683820    Africa Middle Africa          Chad
## 157   25656486  6865154536    Africa Middle Africa           DRC
## 158    1749859  1484579175    Africa Middle Africa   Congo, Rep.
## 159     215014          NA    Africa Middle Africa    Eq. Guinea
## 160     711544  3504843825    Africa Middle Africa         Gabon
## 161    8211950          NA    Africa Middle Africa        Angola
## 162    8932121  6338843529    Africa Middle Africa      Cameroon
```

```
## 163    2274095   735280403    Africa Middle Africa           CAR
## 164    4512795   664885432    Africa Middle Africa          Chad
## 165   26357407  7015838700    Africa Middle Africa           DRC
## 166    1801688  1746408548    Africa Middle Africa   Congo, Rep.
## 167     220605          NA    Africa Middle Africa    Eq. Guinea
## 168     729165  3594320286    Africa Middle Africa         Gabon
## 169    8497950          NA    Africa Middle Africa        Angola
## 170    9201146  7421688027    Africa Middle Africa      Cameroon
## 171    2340259   723923942    Africa Middle Africa           CAR
## 172    4610964   671819643    Africa Middle Africa          Chad
## 173   27049145  7180747678    Africa Middle Africa           DRC
## 174    1855391  2054120887    Africa Middle Africa   Congo, Rep.
## 175     232934          NA    Africa Middle Africa    Eq. Guinea
## 176     747593  3777462740    Africa Middle Africa         Gabon
## 177    8807511          NA    Africa Middle Africa        Angola
## 178    9480638  7979517136    Africa Middle Africa      Cameroon
## 179    2411693   779779920    Africa Middle Africa           CAR
## 180    4716073   707739212    Africa Middle Africa          Chad
## 181   27741104  7147883004    Africa Middle Africa           DRC
## 182    1910800  2538846171    Africa Middle Africa   Congo, Rep.
## 183     251301          NA    Africa Middle Africa    Eq. Guinea
## 184     766867  3660455061    Africa Middle Africa         Gabon
## 185    9128655          NA    Africa Middle Africa        Angola
## 186    9770555  8527457057    Africa Middle Africa      Cameroon
## 187    2485666   716411512    Africa Middle Africa           CAR
## 188    4830055   818703646    Africa Middle Africa          Chad
## 189   28448122  7248789932    Africa Middle Africa           DRC
## 190    1967596  2687469282    Africa Middle Africa   Congo, Rep.
## 191     273199          NA    Africa Middle Africa    Eq. Guinea
## 192     787017  3865742191    Africa Middle Africa         Gabon
## 193    9444918          NA    Africa Middle Africa        Angola
## 194   10070779  9164848021    Africa Middle Africa      Cameroon
## 195    2558432   784338464    Africa Middle Africa           CAR
## 196    4955088   835478056    Africa Middle Africa          Chad
## 197   29190679  7650450747    Africa Middle Africa           DRC
## 198    2025320  2874950661    Africa Middle Africa   Congo, Rep.
## 199     295090          NA    Africa Middle Africa    Eq. Guinea
## 200     808088  4156016995    Africa Middle Africa         Gabon
## 201    9745209  7218737860    Africa Middle Africa        Angola
## 202   10381098  9903824534    Africa Middle Africa      Cameroon
## 203    2627424   815141623    Africa Middle Africa           CAR
## 204    5092650  1017551897    Africa Middle Africa          Chad
## 205   29985665  7686243454    Africa Middle Africa           DRC
## 206    2083648  2840863497    Africa Middle Africa   Congo, Rep.
## 207     314407   194293741    Africa Middle Africa    Eq. Guinea
## 208     830091  4059059225    Africa Middle Africa         Gabon
## 209   10023700  7420862520    Africa Middle Africa        Angola
## 210   10701458 10574478164    Africa Middle Africa      Cameroon
## 211    2691312   844307120    Africa Middle Africa           CAR
## 212    5244158   976018811    Africa Middle Africa          Chad
## 213   30829103  8048819672    Africa Middle Africa           DRC
## 214    2142529  2645936897    Africa Middle Africa   Congo, Rep.
## 215     330247   189765616    Africa Middle Africa    Eq. Guinea
## 216     853039  4026440716    Africa Middle Africa         Gabon
```

```
## 217   10285712  8007110659    Africa Middle Africa        Angola
## 218   11031515 10347481106    Africa Middle Africa      Cameroon
## 219    2751163   802606511    Africa Middle Africa           CAR
## 220    5409275   952703591    Africa Middle Africa          Chad
## 221   31721541  8264177308    Africa Middle Africa           DRC
## 222    2202106  2650947683    Africa Middle Africa   Congo, Rep.
## 223     343290   198185018    Africa Middle Africa    Eq. Guinea
## 224     876877  3336065492    Africa Middle Africa         Gabon
## 225   10544904  8455508856    Africa Middle Africa        Angola
## 226   11370394  9537932265    Africa Middle Africa      Cameroon
## 227    2809720   816333273    Africa Middle Africa           CAR
## 228    5585528  1100204582    Africa Middle Africa          Chad
## 229   32688708  8303050456    Africa Middle Africa           DRC
## 230    2262496  2697770403    Africa Middle Africa   Congo, Rep.
## 231     354488   203447350    Africa Middle Africa    Eq. Guinea
## 232     901473  3764594888    Africa Middle Africa         Gabon
## 233   10820992  8489330892    Africa Middle Africa        Angola
## 234   11716975  9364425783    Africa Middle Africa      Cameroon
## 235    2871005   832474750    Africa Middle Africa           CAR
## 236    5769273  1153946661    Africa Middle Africa          Chad
## 237   33763056  8197929633    Africa Middle Africa           DRC
## 238    2323890  2767909102    Africa Middle Africa   Congo, Rep.
## 239     365451   200946619    Africa Middle Africa    Eq. Guinea
## 240     926648  4086291192    Africa Middle Africa         Gabon
## 241   11127870  8463862899    Africa Middle Africa        Angola
## 242   12070359  8792662258    Africa Middle Africa      Cameroon
## 243    2937832   814596169    Africa Middle Africa           CAR
## 244    5958022  1105729319    Africa Middle Africa          Chad
## 245   34962676  7659464144    Africa Middle Africa           DRC
## 246    2386467  2795588292    Africa Middle Africa   Congo, Rep.
## 247     377363   207499790    Africa Middle Africa    Eq. Guinea
## 248     952269  4298461126    Africa Middle Africa         Gabon
## 249   11472173  8362296544    Africa Middle Africa        Angola
## 250   12430311  8457784979    Africa Middle Africa      Cameroon
## 251    3010950   810096120    Africa Middle Africa           CAR
## 252    6151213  1200104833    Africa Middle Africa          Chad
## 253   36309209  7014456724    Africa Middle Africa           DRC
## 254    2450125  2862682411    Africa Middle Africa   Congo, Rep.
## 255     390381   205141004    Africa Middle Africa    Eq. Guinea
## 256     978252  4561204712    Africa Middle Africa         Gabon
## 257   11848971  7785298083    Africa Middle Africa        Angola
## 258   12796739  8195593373    Africa Middle Africa      Cameroon
## 259    3089141   758054781    Africa Middle Africa           CAR
## 260    6350174  1296130541    Africa Middle Africa          Chad
## 261   37783835  6277938167    Africa Middle Africa           DRC
## 262    2514907  2937112153    Africa Middle Africa   Congo, Rep.
## 263     404081   227081762    Africa Middle Africa    Eq. Guinea
## 264    1004598  4420256843    Africa Middle Africa         Gabon
## 265   12246786  5862329456    Africa Middle Africa        Angola
## 266   13169100  7933334602    Africa Middle Africa      Cameroon
## 267    3170848   760595370    Africa Middle Africa           CAR
## 268    6556628  1092510545    Africa Middle Africa          Chad
## 269   39314955  5432359503    Africa Middle Africa           DRC
## 270    2581306  2907741032    Africa Middle Africa   Congo, Rep.
```

```
## 271     418409   241379316    Africa Middle Africa    Eq. Guinea
## 272    1031358  4594704711    Africa Middle Africa         Gabon
## 273   12648483  6067510987    Africa Middle Africa        Angola
## 274   13546823  7735001163    Africa Middle Africa      Cameroon
## 275    3253698   797864536    Africa Middle Africa           CAR
## 276    6773104  1203257112    Africa Middle Africa          Chad
## 277   40804011  5220497656    Africa Middle Africa           DRC
## 278    2649964  2747815275    Africa Middle Africa   Congo, Rep.
## 279     433197   253727370    Africa Middle Africa    Eq. Guinea
## 280    1058625  4765294832    Africa Middle Africa         Gabon
## 281   13042666  6698532130    Africa Middle Africa        Angola
## 282   13929575  7990255903    Africa Middle Africa      Cameroon
## 283    3335840   855310801    Africa Middle Africa           CAR
## 284    7001634  1218135979    Africa Middle Africa          Chad
## 285   42183620  5257041078    Africa Middle Africa           DRC
## 286    2721277  2857727886    Africa Middle Africa   Congo, Rep.
## 287     448332   289915456    Africa Middle Africa    Eq. Guinea
## 288    1086449  5002313343    Africa Middle Africa         Gabon
## 289   13424813  7448767728    Africa Middle Africa        Angola
## 290   14317191  8389769052    Africa Middle Africa      Cameroon
## 291    3417163   821098357    Africa Middle Africa           CAR
## 292    7242018  1245111272    Africa Middle Africa          Chad
## 293   43424997  5203252472    Africa Middle Africa           DRC
## 294    2795903  2980610185    Africa Middle Africa   Congo, Rep.
## 295     463844   374402170    Africa Middle Africa    Eq. Guinea
## 296    1114879  5183649654    Africa Middle Africa         Gabon
## 297   13801868  8037220379    Africa Middle Africa        Angola
## 298   14709961  8817647112    Africa Middle Africa      Cameroon
## 299    3497910   864616592    Africa Middle Africa           CAR
## 300    7494143  1315502297    Africa Middle Africa          Chad
## 301   44558347  4910983356    Africa Middle Africa           DRC
## 302    2873638  2962726524    Africa Middle Africa   Congo, Rep.
## 303     479836   640931532    Africa Middle Africa    Eq. Guinea
## 304    1143838  5481106515    Africa Middle Africa         Gabon
## 305   14187710  8584134106    Africa Middle Africa        Angola
## 306   15108630  9261990653    Africa Middle Africa      Cameroon
## 307    3577028   905253519    Africa Middle Africa           CAR
## 308    7760157  1406950491    Africa Middle Africa          Chad
## 309   45647949  4831221421    Africa Middle Africa           DRC
## 310    2953011  3072347405    Africa Middle Africa   Congo, Rep.
## 311     496330   781365388    Africa Middle Africa    Eq. Guinea
## 312    1173114  5671730279    Africa Middle Africa         Gabon
## 313   14601983  8862242855    Africa Middle Africa        Angola
## 314   15514249  9668944996    Africa Middle Africa      Cameroon
## 315    3653310   937842672    Africa Middle Africa           CAR
## 316    8042713  1397343126    Africa Middle Africa          Chad
## 317   46788238  4624921463    Africa Middle Africa           DRC
## 318    3031969  2992466373    Africa Middle Africa   Congo, Rep.
## 319     513347  1105213630    Africa Middle Africa    Eq. Guinea
## 320    1202412  5165096006    Africa Middle Africa         Gabon
## 321   15058638  9129180361    Africa Middle Africa        Angola
## 322   15927713 10075040331    Africa Middle Africa      Cameroon
## 323    3726048   959413051    Africa Middle Africa           CAR
## 324    8343321  1385050964    Africa Middle Africa          Chad
```

```
## 325   48048664  4305797176    Africa Middle Africa          DRC
## 326    3109269  3219893817    Africa Middle Africa   Congo, Rep.
## 327     530896  1254223037    Africa Middle Africa    Eq. Guinea
## 328    1231548  5067838984    Africa Middle Africa         Gabon
## 329   15562791  9416016124    Africa Middle Africa        Angola
## 330   16349364 10529854963    Africa Middle Africa      Cameroon
## 331    3794677   961916949    Africa Middle Africa           CAR
## 332    8663599  1546522070    Africa Middle Africa          Chad
## 333   49449015  4215380704    Africa Middle Africa           DRC
## 334    3183883  3342249782    Africa Middle Africa   Congo, Rep.
## 335     549007  2030554290    Africa Middle Africa    Eq. Guinea
## 336    1260435  5175869092    Africa Middle Africa         Gabon
## 337   16109696 10780448534    Africa Middle Africa        Angola
## 338   16779434 10952001542    Africa Middle Africa      Cameroon
## 339    3859784   956312926    Africa Middle Africa           CAR
## 340    9002102  1677840504    Africa Middle Africa          Chad
## 341   50971407  4361586318    Africa Middle Africa           DRC
## 342    3256867  3495993272    Africa Middle Africa   Congo, Rep.
## 343     567664  2425757701    Africa Middle Africa    Eq. Guinea
## 344    1289192  5162041537    Africa Middle Africa         Gabon
## 345   16691395 11137095732    Africa Middle Africa        Angola
## 346   17218591 11393475992    Africa Middle Africa      Cameroon
## 347    3923294   883633144    Africa Middle Africa           CAR
## 348    9353516  1924846596    Africa Middle Africa          Chad
## 349   52602208  4614184091    Africa Middle Africa           DRC
## 350    3331564  3523961218    Africa Middle Africa   Congo, Rep.
## 351     586772  2764278260    Africa Middle Africa    Eq. Guinea
## 352    1318093  5289811986    Africa Middle Africa         Gabon
## 353   17295500 12382535739    Africa Middle Africa        Angola
## 354   17667576 11815245845    Africa Middle Africa      Cameroon
## 355    3987896   892469476    Africa Middle Africa           CAR
## 356    9710498  2572160416    Africa Middle Africa          Chad
## 357   54314855  4920560759    Africa Middle Africa           DRC
## 358    3412592  3647299861    Africa Middle Africa   Congo, Rep.
## 359     606201  3814668806    Africa Middle Africa    Eq. Guinea
## 360    1347524  5361013681    Africa Middle Africa         Gabon
## 361   17912942 14643778507    Africa Middle Africa        Angola
## 362   18126999 12086601214    Africa Middle Africa      Cameroon
## 363    4055608   913888743    Africa Middle Africa           CAR
## 364   10067932  3017980987    Africa Middle Africa          Chad
## 365   56089536  5303776065    Africa Middle Africa           DRC
## 366    3503086  3931789250    Africa Middle Africa   Congo, Rep.
## 367     625866  4186550477    Africa Middle Africa    Eq. Guinea
## 368    1377777  5523001848    Africa Middle Africa         Gabon
## 369   18541467 17680168881    Africa Middle Africa        Angola
## 370   18597109 12476049222    Africa Middle Africa      Cameroon
## 371    4127112   948616515    Africa Middle Africa           CAR
## 372   10423616  3024016949    Africa Middle Africa          Chad
## 373   57926840  5599793442    Africa Middle Africa           DRC
## 374    3604595  4173100882    Africa Middle Africa   Congo, Rep.
## 375     645718  4239290429    Africa Middle Africa    Eq. Guinea
## 376    1408920  5588238866    Africa Middle Africa         Gabon
## 377   19183907 21674668197    Africa Middle Africa        Angola
## 378   19078100 12912710945    Africa Middle Africa      Cameroon
```

```
## 379    4202104   983715326    Africa Middle Africa          CAR
## 380   10779504  3030064983    Africa Middle Africa         Chad
## 381   59834875  5950214773    Africa Middle Africa          DRC
## 382    3715665  4106748578    Africa Middle Africa   Congo, Rep.
## 383     665798  5148299307    Africa Middle Africa    Eq. Guinea
## 384    1440902  5898573546    Africa Middle Africa        Gabon
## 385   19842251 24669478552    Africa Middle Africa       Angola
## 386   19570418 13287179562    Africa Middle Africa     Cameroon
## 387    4280405  1003389633    Africa Middle Africa          CAR
## 388   11139740  3017944723    Africa Middle Africa         Chad
## 389   61809278  6316453096    Africa Middle Africa          DRC
## 390    3832771  4335494474    Africa Middle Africa   Congo, Rep.
## 391     686223  5697997258    Africa Middle Africa    Eq. Guinea
## 392    1473741  6035623996    Africa Middle Africa        Gabon
## 393   20520103 25264731265    Africa Middle Africa       Angola
## 394   20074522 13552923153    Africa Middle Africa     Cameroon
## 395    4361492  1020447256    Africa Middle Africa          CAR
## 396   11510535  2981729387    Africa Middle Africa         Chad
## 397   63845097  6495485661    Africa Middle Africa          DRC
## 398    3950786  4659307269    Africa Middle Africa   Congo, Rep.
## 399     707155  6024744206    Africa Middle Africa    Eq. Guinea
## 400    1507428  5950756011    Africa Middle Africa        Gabon
## 401   21219954 26125663270    Africa Middle Africa       Angola
## 402   20590666 13986616694    Africa Middle Africa     Cameroon
## 403    4444973  1054122016    Africa Middle Africa          CAR
## 404   11896380  3369354207    Africa Middle Africa         Chad
## 405   65938712  6961485000    Africa Middle Africa          DRC
## 406    4066078  5067059617    Africa Middle Africa   Congo, Rep.
## 407     728710  5979285835    Africa Middle Africa    Eq. Guinea
## 408    1541936  6343809583    Africa Middle Africa        Gabon
## 409   21942296 27013935821    Africa Middle Africa       Angola
## 410   21119065 14518108128    Africa Middle Africa     Cameroon
## 411    4530903  1086799798    Africa Middle Africa          CAR
## 412   12298512  3473804187    Africa Middle Africa         Chad
## 413   68087376  7440411972    Africa Middle Africa          DRC
## 414    4177435  5293029727    Africa Middle Africa   Congo, Rep.
## 415     750918  6403432391    Africa Middle Africa    Eq. Guinea
## 416    1577298  6649166796    Africa Middle Africa        Gabon
## 417   22685632          NA    Africa Middle Africa       Angola
## 418   21659488          NA    Africa Middle Africa     Cameroon
## 419    4619500          NA    Africa Middle Africa          CAR
## 420   12715465          NA    Africa Middle Africa         Chad
## 421   70291160          NA    Africa Middle Africa          DRC
## 422    4286188          NA    Africa Middle Africa   Congo, Rep.
## 423     773729          NA    Africa Middle Africa    Eq. Guinea
## 424    1613489          NA    Africa Middle Africa        Gabon
## 425   23448202          NA    Africa Middle Africa       Angola
## 426   22211166          NA    Africa Middle Africa     Cameroon
## 427    4710678          NA    Africa Middle Africa          CAR
## 428   13145788          NA    Africa Middle Africa         Chad
## 429   72552861          NA    Africa Middle Africa          DRC
## 430    4394334          NA    Africa Middle Africa   Congo, Rep.
## 431     797082          NA    Africa Middle Africa    Eq. Guinea
## 432    1650351          NA    Africa Middle Africa        Gabon
```

```
## 433   24227524          NA    Africa Middle Africa        Angola
## 434   22773014          NA    Africa Middle Africa      Cameroon
## 435    4804316          NA    Africa Middle Africa           CAR
## 436   13587053          NA    Africa Middle Africa          Chad
## 437   74877030          NA    Africa Middle Africa           DRC
## 438    4504962          NA    Africa Middle Africa   Congo, Rep.
## 439     820885          NA    Africa Middle Africa    Eq. Guinea
## 440    1687673          NA    Africa Middle Africa         Gabon
## 441   25021974          NA    Africa Middle Africa        Angola
## 442   23344179          NA    Africa Middle Africa      Cameroon
## 443    4900274          NA    Africa Middle Africa           CAR
## 444   14037472          NA    Africa Middle Africa          Chad
## 445   77266814          NA    Africa Middle Africa           DRC
## 446    4620330          NA    Africa Middle Africa   Congo, Rep.
## 447     845060          NA    Africa Middle Africa    Eq. Guinea
## 448    1725292          NA    Africa Middle Africa         Gabon
## 449         NA          NA    Africa Middle Africa        Angola
## 450         NA          NA    Africa Middle Africa      Cameroon
## 451         NA          NA    Africa Middle Africa           CAR
## 452         NA          NA    Africa Middle Africa          Chad
## 453         NA          NA    Africa Middle Africa           DRC
## 454         NA          NA    Africa Middle Africa   Congo, Rep.
## 455         NA          NA    Africa Middle Africa    Eq. Guinea
## 456         NA          NA    Africa Middle Africa         Gabon
```

☐ A.

```
dat <- gapminder %>% filter(region == "Middle Africa") %>%
  mutate(recode(country,
                         "Central African Republic" = "CAR",
                         "Congo, Dem. Rep." = "DRC",
                         "Equatorial Guinea" = "Eq. Guinea"))
```

☐ B.

```
dat <- gapminder %>% filter(region == "Middle Africa") %>%
  mutate(country_short = recode(country,
                         c("Central African Republic", "Congo, Dem. Rep.", "Equatorial Guinea"),
                         c("CAR", "DRC", "Eq. Guinea")))
```

☐ C.

```
dat <- gapminder %>% filter(region == "Middle Africa") %>%
  mutate(country = recode(country,
                         "Central African Republic" = "CAR",
                         "Congo, Dem. Rep." = "DRC",
                         "Equatorial Guinea" = "Eq. Guinea"))
```

☒ D.

```r
dat <- gapminder %>% filter(region == "Middle Africa") %>%
  mutate(country_short = recode(country,
                          "Central African Republic" = "CAR",
                          "Congo, Dem. Rep." = "DRC",
                          "Equatorial Guinea" = "Eq. Guinea"))
```

5. Import raw Brexit referendum polling data from Wikipedia:

```r
if(!require(stringr)) install.packages("stringr")

library(stringr)
url <- "https://en.wikipedia.org/w/index.php?title=Opinion_polling_for_the_United_Kingdom_European_Unio
tab <- read_html(url) %>% html_nodes("table")
polls <- tab[[5]] %>% html_table(fill = TRUE)
```

You will use a variety of string processing techniques learned in this section to reformat these data.

Some rows in this table do not contain polls. You can identify these by the lack of the percent sign (%) in the Remain column.

Update `polls` by changing the column names to `c("dates", "remain", "leave", "undecided", "lead", "samplesize", "pollster", "poll_type", "notes")` and only keeping rows that have a percent sign (%) in the remain column.

How many rows remain in the `polls` data frame?

```r
names(polls) <- c("dates", "remain", "leave", "undecided", "lead", "samplesize", "pollster", "poll_type
polls <- polls[str_detect(polls$remain, "%"), -9]
nrow(polls)
```

```
## [1] 129
```

6. The `remain` and `leave` columns are both given in the format "48.1%": percentages out of 100% with a percent symbol.

Which of these commands converts the `remain` vector to a proportion between 0 and 1?

Check all correct answers.

- ☐ A. `as.numeric(str_remove(polls$remain, "%"))`
- ☐ B. `as.numeric(polls$remain)/100`
- ☐ C. `parse_number(polls$remain)`
- ☐ D. `str_remove(polls$remain, "%")/100`
- ☒ E. `as.numeric(str_replace(polls$remain, "%", ""))/100`
- ☒ F. `parse_number(polls$remain)/100`

7. The `undecided` column has some "N/A" values. These "N/A"s are only present when the `remain` and `leave` columns total 100%, so they should actually be zeros.

Use a function from **stringr** to convert "N/A" in the `undecided` column to 0. The format of your command should be `function_name(polls$undecided, "arg1", "arg2")`.

What function replaces `function_name`? `str_replace`

What argument replaces `arg1`? `N/A`

What argument replaces `arg2`? `0`

8. The `dates` column contains the range of dates over which the poll was conducted. The format is “8-10 Jan” where the poll had a start date of 2016-01-08 and end date of 2016-01-10. Some polls go across month boundaries (16 May-12 June).

The end date of the poll will always be one or two digits, followed by a space, followed by the month as one or more letters (either capital or lowercase). In these data, all month abbreviations or names have 3, 4 or 5 letters.

Write a regular expression to extract the end day and month from `dates`. Insert it into the skeleton code below:

```
temp <- str_extract_all(polls$dates, _____)
end_date <- sapply(temp, function(x) x[length(x)]) # take last element (handles polls that cross month
```

Which of the following regular expressions correctly extracts the end day and month when inserted into the blank in the code above? Check all correct answers.

- ☐ A. `"\\d?\\s[a-zA-Z]?"`
- ☒ B. `"\\d+\\s[a-zA-Z]+"`
- ☐ C. `"\\d+\\s[A-Z]+"`
- ☒ D. `"[0-9]+\\s[a-zA-Z]+"`
- ☒ E. `"\\d{1,2}\\s[a-zA-Z]+"`
- ☐ F. `"\\d{1,2}[a-zA-Z]+"`
- ☒ G. `"\\d+\\s[a-zA-Z]{3,5}"`

Section 4 Overview

In the **Dates, Times, and Text Mining** section, you will learn how to deal with dates and times in R and also how to generate numerical summaries from text data.

After completing this section, you will be able to:

- Handle **dates** and **times** in R.
- Use the **lubridate** package to parse dates and times in different formats.
- Generate **numerical summaries from text data** and apply data visualization and analysis techniques to those data.

Dates and Times

The textbook for this section is available here.

Key points

- Dates are a separate data type in R.The **tidyverse** includes functionality for dealing with dates through the **lubridate** package.
- Extract the year, month and day from a date object with the `year()`, `month()` and `day()` functions.
- Parsers convert strings into dates with the standard YYYY-MM-DD format (ISO 8601 format). Use the parser with the name corresponding to the string format of year, month and day (`ymd()`, `ydm()`, `myd()`, `mdy()`, `dmy()`, `dym()`).
- Get the current time with the `Sys.time()` function. Use the `now()` function instead to specify a time zone.
- You can extract values from time objects with the `hour()`, `minute()` and `second()` functions.

- Parsers convert strings into times (for example, `hms()`). Parsers can also create combined date-time objects (for example, `mdy_hms()`).

Code

```
# inspect the startdate column of 2016 polls data, a Date type
data("polls_us_election_2016")
polls_us_election_2016$startdate %>% head
```

```
## [1] "2016-11-03" "2016-11-01" "2016-11-02" "2016-11-04" "2016-11-03"
## [6] "2016-11-03"
```

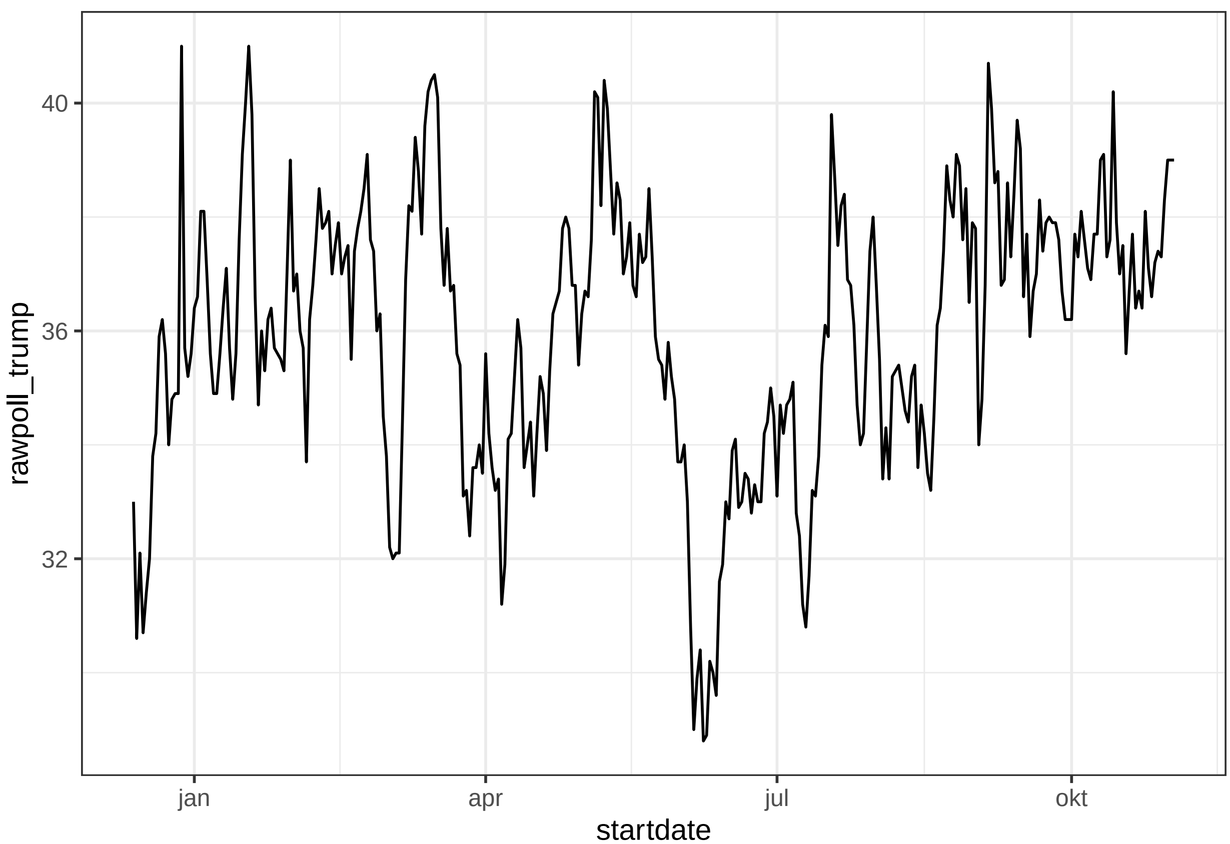
```
class(polls_us_election_2016$startdate)
```

```
## [1] "Date"
```

```
as.numeric(polls_us_election_2016$startdate) %>% head
```

```
## [1] 17108 17106 17107 17109 17108 17108
```

```
# ggplot is aware of dates
polls_us_election_2016 %>% filter(pollster == "Ipsos" & state =="U.S.") %>%
  ggplot(aes(startdate, rawpoll_trump)) +
  geom_line()
```



```r
# lubridate: the tidyverse date package
if(!require(lubridate)) install.packages("lubridate")
```

```
## Loading required package: lubridate

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
library(lubridate)

# select some random dates from polls
set.seed(2)
dates <- sample(polls_us_election_2016$startdate, 10) %>% sort
dates
```

```
##  [1] "2016-01-19" "2016-08-06" "2016-08-26" "2016-09-09" "2016-09-14"
##  [6] "2016-09-16" "2016-09-29" "2016-10-04" "2016-10-12" "2016-10-23"
```

```r
# extract month, day, year from date strings
data.frame(date = dates,
           month = month(dates),
           day = day(dates),
           year = year(dates))
```

```
##          date month day year
## 1  2016-01-19     1  19 2016
## 2  2016-08-06     8   6 2016
## 3  2016-08-26     8  26 2016
## 4  2016-09-09     9   9 2016
## 5  2016-09-14     9  14 2016
## 6  2016-09-16     9  16 2016
## 7  2016-09-29     9  29 2016
## 8  2016-10-04    10   4 2016
## 9  2016-10-12    10  12 2016
## 10 2016-10-23    10  23 2016
```

```r
month(dates, label = TRUE)    # extract month label
```

```
##  [1] jan aug aug sep sep sep sep okt okt okt
## 12 Levels: jan < feb < mrt < apr < mei < jun < jul < aug < sep < ... < dec
```

```r
# ymd works on mixed date styles
x <- c(20090101, "2009-01-02", "2009 01 03", "2009-1-4",
       "2009-1, 5", "Created on 2009 1 6", "200901 !!! 07")
ymd(x)
```

```
## [1] "2009-01-01" "2009-01-02" "2009-01-03" "2009-01-04" "2009-01-05"
## [6] "2009-01-06" "2009-01-07"
```

```r
# different parsers extract year, month and day in different orders
x <- "09/01/02"
ymd(x)
```

```
## [1] "2009-01-02"
```

```r
mdy(x)
```

```
## [1] "2002-09-01"
```

```r
ydm(x)
```

```
## [1] "2009-02-01"
```

```r
myd(x)
```

```
## [1] "2001-09-02"
```

```r
dmy(x)
```

```
## [1] "2002-01-09"
```

```r
dym(x)
```

```
## [1] "2001-02-09"
```

```r
now()    # current time in your time zone
```

```
## [1] "2020-09-11 17:31:38 CEST"
```

```r
now("GMT")    # current time in GMT
```

```
## [1] "2020-09-11 15:31:38 GMT"
```

```r
now() %>% hour()    # current hour
```

```
## [1] 17
```

```r
now() %>% minute()    # current minute
```

```
## [1] 31
```

```r
now() %>% second()    # current second
```

```
## [1] 38.60838
```

```r
# parse time
x <- c("12:34:56")
hms(x)
```

```
## [1] "12H 34M 56S"
```

```r
#parse datetime
x <- "Nov/2/2012 12:34:56"
mdy_hms(x)
```

```
## [1] "2012-11-02 12:34:56 UTC"
```

Text mining

The textbook for this section is available here.

Key points

- The **tidytext** package helps us convert free form text into a tidy table.
- Use `unnest_tokens()` to extract individual words and other meaningful chunks of text.
- Sentiment analysis assigns emotions or a positive/negative score to tokens. You can extract sentiments using `get_sentiments()`. Common lexicons for sentiment analysis are "bing", "afinn", "nrc" and "loughran".

With the exception of labels used to represent categorical data, we have focused on numerical data, but in many applications data starts as text. Well known examples are spam filtering, cyber-crime prevention, counter-terrorism and sentiment analysis.

In all these examples, the raw data is composed of free form texts. Our task is to extract insights from these data. In this section, we learn how to generate useful numerical summaries from text data to which we can apply some of the powerful data visualization and analysis techniques we have learned.

Case study: Trump Tweets

See my GitHub-repository on Trump Tweets.

Assessment Part 1 - Dates, Times, and Text Mining

```r
options(digits = 3)    # 3 significant digits
```

1. Which of the following is the standard ISO 8601 format for dates?

☐ A. MM-DD-YY
☒ B. YYYY-MM-DD
☐ C. YYYYMMDD
☐ D. YY-MM-DD

2. Which of the following commands could convert this string into the correct date format?

```
dates <- c("09-01-02", "01-12-07", "02-03-04")
```

- ☐ A. `ymd(dates)`
- ☐ B. `mdy(dates)`
- ☐ C. `dmy(dates)`
- ☒ D. It is impossible to know which format is correct without additional information.

3. Load the `brexit_polls` data frame from dslabs:

```
data(brexit_polls)
```

How many polls had a start date (startdate) in April (month number 4)?

```
sum(month(brexit_polls$startdate) == 4)
```

```
## [1] 25
```

Use the `round_date()` function on the `enddate` column with the argument `unit="week"`. How many polls ended the week of 2016-06-12?

Read the documentation to learn more about `round_date()`.

```
sum(round_date(brexit_polls$enddate, unit = "week") == "2016-06-12")
```

```
## [1] 13
```

4. Use the `weekdays()` function from **lubridate** to determine the weekday on which each poll ended (`enddate`).

On which weekday did the greatest number of polls end?

```
table(weekdays(brexit_polls$enddate))
```

```
##
##    dinsdag donderdag    maandag    vrijdag   woensdag   zaterdag     zondag
##         23        17         20         14         12          4         37
```

- ☐ A. Monday
- ☐ B. Tuesday
- ☐ C. Wednesday
- ☐ D. Thursday
- ☐ E. Friday
- ☐ F. Saturday
- ☒ G. Sunday

```r
max(weekdays(brexit_polls$enddate))
```

```
## [1] "zondag"
```

5. Load the `movielens` data frame from **dslabs**.

```r
data(movielens)
```

This data frame contains a set of about 100,000 movie reviews. The `timestamp` column contains the review date as the number of seconds since 1970-01-01 (epoch time).

Convert the `timestamp` column to dates using the **lubridate** as_datetime() function.

Which year had the most movie reviews?

```r
dates <- as_datetime(movielens$timestamp)
reviews_by_year <- table(year(dates))
names(which.max(reviews_by_year))
```

```
## [1] "2000"
```

Which hour of the day had the most movie reviews?

```r
reviews_by_hour <- table(hour(dates))
names(which.max(reviews_by_hour))
```

```
## [1] "20"
```

Assessment Part 2 - Dates, Times, and Text Mining

6. Project Gutenberg is a digital archive of public domain books. The R package **gutenbergr** facilitates the importation of these texts into R. We will combine this with the **tidyverse** and **tidytext** libraries to practice text mining.

Use these libraries and options:

```r
if(!require(gutenbergr)) install.packages("gutenbergr")
```

```
## Loading required package: gutenbergr
```

```r
if(!require(tidytext)) install.packages("tidytext")
```

```
## Loading required package: tidytext
```

```r
library(gutenbergr)
library(tidytext)
```

You can see the books and documents available in **gutenbergr** like this:

```r
gutenberg_metadata
```

```
## # A tibble: 51,997 x 8
##    gutenberg_id title author gutenberg_autho~ language gutenberg_books~ rights
##           <int> <chr> <chr>             <int> <chr>    <chr>            <chr>
##  1            0  <NA> <NA>                 NA en       <NA>             Publi~
##  2            1 "The~ Jeffe~             1638 en       United States L~ Publi~
##  3            2 "The~ Unite~                1 en       American Revolu~ Publi~
##  4            3 "Joh~ Kenne~             1666 en       <NA>             Publi~
##  5            4 "Lin~ Linco~                3 en       US Civil War     Publi~
##  6            5 "The~ Unite~                1 en       American Revolu~ Publi~
##  7            6 "Giv~ Henry~                4 en       American Revolu~ Publi~
##  8            7 "The~ <NA>                 NA en       <NA>             Publi~
##  9            8 "Abr~ Linco~                3 en       US Civil War     Publi~
## 10            9 "Abr~ Linco~                3 en       US Civil War     Publi~
## # ... with 51,987 more rows, and 1 more variable: has_text <lgl>
```

Use `str_detect()` to find the ID of the novel *Pride and Prejudice*.

How many different ID numbers are returned?

```r
gutenberg_metadata %>%
    filter(str_detect(title, "Pride and Prejudice"))
```

```
## # A tibble: 6 x 8
##   gutenberg_id title author gutenberg_autho~ language gutenberg_books~ rights
##          <int> <chr> <chr>             <int> <chr>    <chr>            <chr>
## 1         1342 Prid~ Auste~               68 en       Best Books Ever~ Publi~
## 2        20686 Prid~ Auste~               68 en       Harvard Classic~ Publi~
## 3        20687 Prid~ Auste~               68 en       Harvard Classic~ Publi~
## 4        26301 Prid~ Auste~               68 en       Best Books Ever~ Publi~
## 5        37431 Prid~ <NA>                 NA en       <NA>             Publi~
## 6        42671 Prid~ Auste~               68 en       Best Books Ever~ Publi~
## # ... with 1 more variable: has_text <lgl>
```

7. Notice that there are several versions of the book. The `gutenberg_works()` function filters this table to remove replicates and include only English language works. Use this function to find the ID for *Pride and Prejudice*.

What is the correct ID number?

Read the `gutenberg_works()` documentation to learn how to use the function.

```r
gutenberg_works(title == "Pride and Prejudice")$gutenberg_id
```

```
## Warning: `filter_()` is deprecated as of dplyr 0.7.0.
## Please use `filter()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
## Warning: `distinct_()` is deprecated as of dplyr 0.7.0.
## Please use `distinct()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
## [1] 1342
```

8. Use the `gutenberg_download()` function to download the text for Pride and Prejudice. Use the **tidytext** package to create a tidy table with all the words in the text. Save this object as '`words`.

How many words are present in the book?

```
book <- gutenberg_download(1342)
```

```
## Determining mirror for Project Gutenberg from http://www.gutenberg.org/robot/harvest
```

```
## Using mirror http://aleph.gutenberg.org
```

```
words <- book %>%
  unnest_tokens(word, text)
nrow(words)
```

```
## [1] 122204
```

9. Remove stop words from the `words` object. Recall that stop words are defined in the `stop_words` data frame from the **tidytext** package.

How many words remain?

```
words <- words %>% anti_join(stop_words)
```

```
## Joining, by = "word"
```

```
nrow(words)
```

```
## [1] 37246
```

10. After removing stop words, detect and then filter out any token that contains a digit from '`words`.

How many words remain?

```
words <- words %>%
  filter(!str_detect(word, "\\d"))
nrow(words)
```

```
## [1] 37180
```

11. Analyze the most frequent words in the novel after removing stop words and tokens with digits.

How many words appear more than 100 times in the book?

```
words %>%
    count(word) %>%
    filter(n > 100) %>%
    nrow()
```

```
## [1] 23
```

What is the most common word in the book?

```
words %>%
    count(word) %>%
    top_n(1, n) %>%
    pull(word)
```

```
## [1] "elizabeth"
```

How many times does that most common word appear?

```
words %>%
    count(word) %>%
    top_n(1, n) %>%
    pull(n)
```

```
## [1] 597
```

12. Define the afinn lexicon.

```
afinn <- get_sentiments("afinn")
```

Note that this command will trigger a question in the R Console asking if you want to download the AFINN lexicon. Press 1 to select "Yes" (if using RStudio, enter this in the Console tab).

Use this `afinn` lexicon to assign sentiment values to `words`. Keep only words that are present in both `words` and the `afinn` lexicon. Save this data frame as `afinn_sentiments`.

How many elements of words have sentiments in the afinn lexicon?

```
afinn_sentiments <- inner_join(afinn, words)
```

```
## Joining, by = "word"
```

```
nrow(afinn_sentiments)
```

```
## [1] 6065
```

What proportion of words in afinn_sentiments have a positive value?

```r
mean(afinn_sentiments$value > 0)
```

```
## [1] 0.563
```

How many elements of afinn_sentiments have a value of 4?

```r
sum(afinn_sentiments$value == 4)
```

```
## [1] 51
```

Final: Comprehensive Assessment

Comprehensive Assessment: Puerto Rico Hurricane Mortality

Project Introduction

On September 20, 2017, Hurricane Maria made landfall on Puerto Rico. It was the worst natural disaster on record in Puerto Rico and the deadliest Atlantic hurricane since 2004. However, Puerto Rico's official death statistics only tallied 64 deaths caused directly by the hurricane (due to structural collapse, debris, floods and drownings), an undercount that slowed disaster recovery funding. The majority of the deaths resulted from infrastructure damage that made it difficult to access resources like clean food, water, power, healthcare and communications in the months after the disaster, and although these deaths were due to effects of the hurricane, they were not initially counted.

In order to correct the misconception that few lives were lost in Hurricane Maria, statisticians analyzed how death rates in Puerto Rico changed after the hurricane and estimated the excess number of deaths likely caused by the storm. This analysis suggested that the actual number of deaths in Puerto Rico was 2,975 (95% CI: 2,658-3,290) over the 4 months following the hurricane, much higher than the original count.

We will use your new data wrangling skills to extract actual daily mortality data from Puerto Rico and investigate whether the Hurricane Maria had an immediate effect on daily mortality compared to unaffected days in September 2015-2017.

```r
options(digits = 3)    # report 3 significant digits
```

Puerto Rico Hurricane Mortality - Part 1

1. In the `extdata` directory of the **dslabs** package, you will find a PDF file containing daily mortality data for Puerto Rico from Jan 1, 2015 to May 31, 2018. You can find the file like this:

```r
fn <- system.file("extdata", "RD-Mortality-Report_2015-18-180531.pdf", package="dslabs")
```

Find and open the file or open it directly from RStudio. On a Mac, you can type:

```r
system2("open", args = fn)
```

and on Windows, you can type:

```r
system("cmd.exe", input = paste("start", fn))
```

Which of the following best describes this file?

- ☐ A. It is a table. Extracting the data will be easy.
- ☐ B. It is a report written in prose. Extracting the data will be impossible.
- ☒ C. It is a report combining graphs and tables. Extracting the data seems possible.
- ☐ D. It shows graphs of the data. Extracting the data will be difficult.

2. We are going to create a tidy dataset with each row representing one observation. The variables in this dataset will be year, month, day and deaths.

Use the **pdftools** package to read in `fn` using the `pdf_text` function. Store the results in an object called txt.

```r
txt <- pdf_text(fn)
class(txt)
```

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

```r
str(txt)
```

```
##  chr [1:12] "6/4/2018                          Departamento de Salud - Registro Demográfico - División d
```

```r
length(txt)
```

```
## [1] 12
```

Describe what you see in `txt`.

- ☐ A. A table with the mortality data.
- ☒ B. A character string of length 12. Each entry represents the text in each page. The mortality data is in there somewhere.
- ☐ C. A character string with one entry containing all the information in the PDF file.
- ☐ D. An html document.

3. Extract the ninth page of the PDF file from the object `txt`, then use the `str_split` function from the **stringr** package so that you have each line in a different entry. The new line character is `\n`. Call this string vector `x`.

Look at `x`. What best describes what you see?

What kind of object is `x`?

How many entries does `x` have?

```r
x <- str_split(txt[9], "\n")
class(x)
```

```
## [1] "list"
```

```
length(x)
```

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

- ☐ A. It is an empty string.
- ☐ B. I can see the figure shown in page 1.
- ☐ C. It is a tidy table.
- ☒ D. I can see the table! But there is a bunch of other stuff we need to get rid of.

4. Define `s` to be the first entry of the `x` object.

What kind of object is s?

How many entries does s have?

```
s <- x[[1]]
class(s)
```

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

```
length(s)
```

```
## [1] 40
```

5. When inspecting the string we obtained above, we see a common problem: white space before and after the other characters. Trimming is a common first step in string processing. These extra spaces will eventually make splitting the strings hard so we start by removing them.

We learned about the command `str_trim`that removes spaces at the start or end of the strings. Use this function to trim `s` and assign the result to `s` again.

After trimming, what single character is the last character of element 1 of `s`?

```
s <- str_trim(s)
s[1] # print string, visually inspect last character
```

```
## [1] "6/4/2018                              Departamento de Salud - Registro Demográfico - División d
```

6. We want to extract the numbers from the strings stored in `s`. However, there are a lot of non-numeric characters that will get in the way. We can remove these, but before doing this we want to preserve the string with the column header, which includes the month abbreviation.

Use the `str_which` function to find the row with the header. Save this result to `header_index`.
Hint: find the first string that matches the pattern `"2015"` using the `str_which` function.

What is the value of `header_index`?

```
header_index <- str_which(s, pattern="2015")[1]
header_index
```

```
## [1] 2
```

7. We want to extract two objects from the header row: `month` will store the month and `header` will store the column names.
   Save the content of the header row into an object called `header`, then use `str_split` to help define the two objects we need.

What is the value of month? Use `header_index` to extract the row. The separator here is one or more spaces. Also, consider using the `simplify` argument.

What is the third value in header?

```r
tmp <- str_split(s[header_index], pattern="\\s+", simplify=TRUE)
month <- tmp[1]
header <- tmp[-1]
month
```

```
## [1] "SEP"
```

```r
header[3]
```

```
## [1] "2017"
```

Puerto Rico Hurricane Mortality - Part 2

8. Notice that towards the end of the page defined by `s` you see a *"Total"* row followed by rows with other summary statistics. Create an object called `tail_index` with the index of the *"Total"* entry.

What is the value of `tail_index`?

```r
tail_index <- str_which(s, "Total")
tail_index
```

```
## [1] 35
```

9. Because our PDF page includes graphs with numbers, some of our rows have just one number (from the y-axis of the plot). Use the `str_count` function to create an object `n` with the count of numbers in each row.

How many rows have a single number in them? You can write a regex for a number like this `\\d+`.

```r
n <- str_count(s, "\\d+")
sum(n == 1)
```

```
## [1] 2
```

10. We are now ready to remove entries from rows that we know we don't need. The entry `header_index` and everything before it should be removed. Entries for which `n` is 1 should also be removed, and the entry `tail_index` and everything that comes after it should be removed as well.

How many entries remain in `s`?

```r
out <- c(1:header_index, which(n==1), tail_index:length(s))
s <- s[-out]
length(s)
```

```
## [1] 30
```

11. Now we are ready to remove all text that is not a digit or space. Do this using regular expressions (regex) and the `str_remove_all` function. In regex, using the ^ inside the square brackets `[]` means *not*, like the `!` means *not* in `!=`. To define the regex pattern to catch all non-numbers, you can type `[^\\d]`. But remember you also want to keep spaces.

Which of these commands produces the correct output?

☐ A.

```r
s <- str_remove_all(s, "[^\\d]")
```

☐ B.

```r
s <- str_remove_all(s, "[\\d\\s]")
```

☒ C.

```r
s <- str_remove_all(s, "[^\\d\\s]")
```

☐ D.

```r
s <- str_remove_all(s, "[\\d]")
```

12. Use the `str_split_fixed` function to convert `s` into a data matrix with just the day and death count data:

```r
s <- str_split_fixed(s, "\\s+", n = 6)[,1:5]
```

Now you are almost ready to finish. Add column names to the matrix: the first column should be `day` and the next columns should be the `header`. Convert all values to numeric. Also, add a column with the month. Call the resulting object `tab`.

What was the mean number of deaths per day in September 2015?

```r
tab <- s %>%
    as_data_frame() %>%
    setNames(c("day", header)) %>%
    mutate_all(as.numeric)
mean(tab$"2015")
```

```
## [1] 75.3
```

What is the mean number of deaths per day in September 2016?

```r
mean(tab$`2016`)
```

```
## [1] 78.9
```

Hurricane Maria hit Puerto Rico on September 20, 2017. What was the mean number of deaths per day from September 1-19, 2017, before the hurricane hit?

```r
mean(tab$`2017`[1:19])
```

```
## [1] 83.7
```

What was the mean number of deaths per day from September 20-30, 2017, after the hurricane hit?

```r
mean(tab$`2017`[20:30])
```

```
## [1] 122
```

13. Finish it up by changing `tab` to a tidy format, starting from this code outline:

```r
tab <- tab %>% ________(year, deaths, -day) %>%
    mutate(deaths = as.numeric(deaths))
tab
```

What code fills the blank to generate a data frame with columns named "day", "year" and "deaths"?

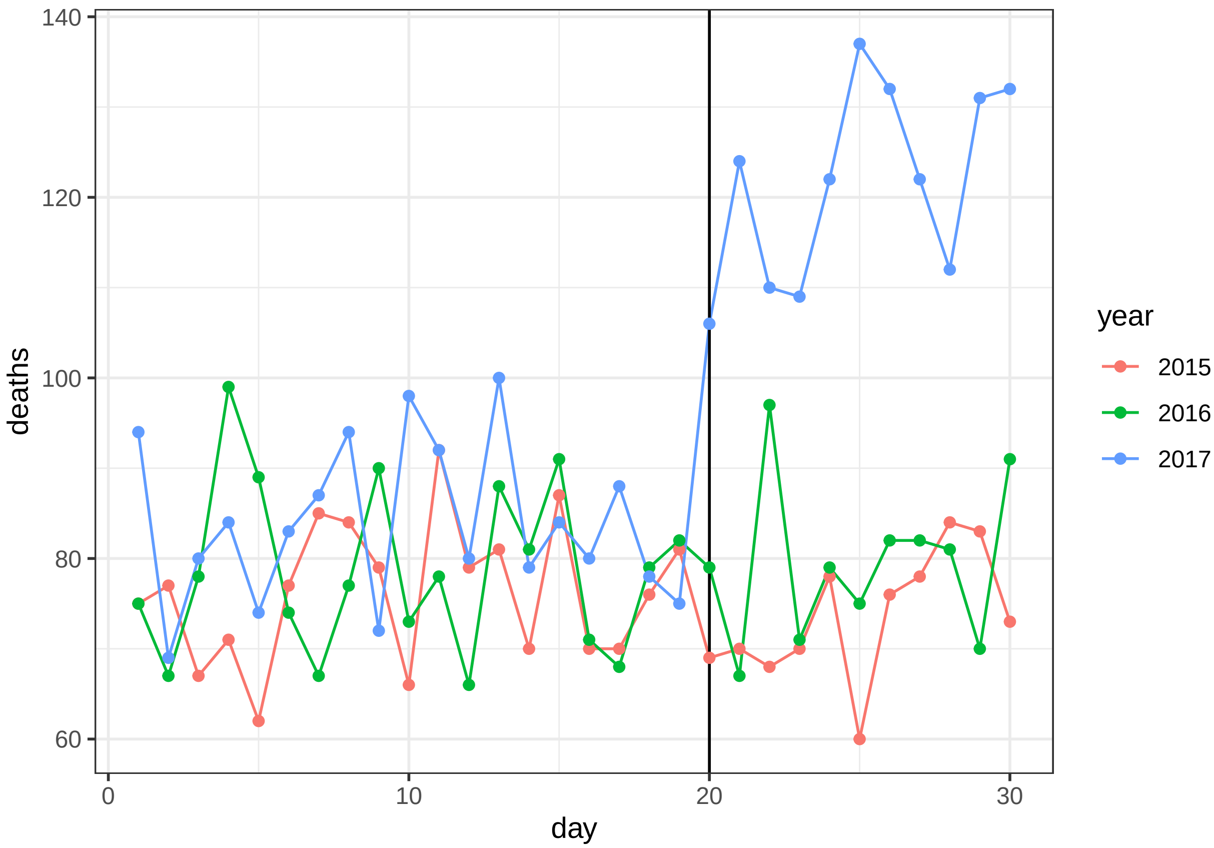
```r
tab <- tab %>% gather(year, deaths, -day) %>%
    mutate(deaths = as.numeric(deaths))
tab
```

```
## # A tibble: 120 x 3
##      day year  deaths
##    <dbl> <chr>  <dbl>
##  1     1 2015      75
##  2     2 2015      77
##  3     3 2015      67
##  4     4 2015      71
##  5     5 2015      62
##  6     6 2015      77
##  7     7 2015      85
##  8     8 2015      84
##  9     9 2015      79
## 10    10 2015      66
## # ... with 110 more rows
```

- ☐ A. separate
- ☐ B. unite
- ☒ C. gather
- ☐ D. spread

14. Make a plot of deaths versus day with color to denote year. Exclude 2018 since we have no data. Add a vertical line at day 20, the day that Hurricane Maria hit in 2017.

```
tab %>% filter(year < 2018) %>%
        ggplot(aes(day, deaths, color = year)) +
        geom_line() +
        geom_vline(xintercept = 20) +
        geom_point()
```



Which of the following are TRUE?

☒ A. September 2015 and 2016 deaths by day are roughly equal to each other.
☐ B. The day with the most deaths was the day of the hurricane: September 20, 2017.
☒ C. After the hurricane in September 2017, there were over 100 deaths per day every day for the rest of the month.
☒ D. No days before September 20, 2017 have over 100 deaths per day.