Chapter 5

Chapter 5 Importing data

We have been using data sets already stored as R objects. A data scientist will rarely have such luck and will have to import data into R from either a file, a database, or other sources. Currently, one of the most common ways of storing and sharing data for analysis is through electronic spreadsheets. A spreadsheet stores data in rows and columns. It is basically a file version of a data frame. When saving such a table to a computer file, one needs a way to define when a new row or column ends and the other begins. This in turn defines the cells in which single values are stored.

When creating spreadsheets with text files, like the ones created with a simple text editor, a new row is defined with return and columns are separated with some predefined special character. The most common characters are comma (,), semicolon (;), space ( ), and tab (a preset number of spaces or). Here is an example of what a comma separated file looks like if we open it with a basic text editor:

The first row contains column names rather than data. We call this a header, and when we read-in data from a spreadsheet it is important to know if the file has a header or not. Most reading functions assume there is a header. To know if the file has a header, it helps to look at the file before trying to read it. This can be done with a text editor or with RStudio. In RStudio, we can do this by either opening the file in the editor or navigating to the file location, double clicking on the file, and hitting View File.

However, not all spreadsheet files are in a text format. Google Sheets, which are rendered on a browser, are an example. Another example is the proprietary format used by Microsoft Excel. These can't be viewed with a text editor. Despite this, due to the widespread use of Microsoft Excel software, this format is widely used.

We start this chapter by describing the difference between text (ASCII), Unicode, and binary files and how this affects how we import them. We then explain the concepts of file paths and working directories, which are essential to understand how to import data effectively. We then introduce the readr and readxl package and the functions that are available to import spreadsheets into R. Finally, we provide some recommendations on how to store and organize data in files. More complex challenges such as extracting data from web pages or PDF documents are left for the Data Wrangling part of the book.

5.1 Paths and the working directory

The first step when importing data from a spreadsheet is to locate the file containing the data. Although we do not recommend it, you can use an approach similar to what you do to open files in Microsoft Excel by clicking on the RStudio "File" menu, clicking "Import Dataset," then clicking through folders until you find the file. We want to be able to write code rather than use the point-and-click approach. The keys and concepts we need to learn to do this are described in detail in the Productivity Tools part of this book. Here we provide an overview of the very basics.

The main challenge in this first step is that we need to let the R functions doing the importing know where to look for the file containing the data. The simplest way to do this is to have a copy of the file in the folder in which the importing functions look by default. Once we do this, all we have to supply to the importing function is the filename.

A spreadsheet containing the US murders data is included as part of the dslabs package. Finding this file is not straightforward, but the following lines of code copy the file to the folder in which R looks in by default. We explain how these lines work below.

```
filename <- "murders.csv"
dir <- system.file("extdata", package="dslabs")
fullpath <- file.path(dir, filename)
file.copy(fullpath, "murders.csv")
```

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

This code does not read the data into R, it just copies a file. But once the file is copied, we can import the data with a simple line of code. Here we use the read_csv function from the readr package, which is part of the tidyverse.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1
```

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

```
dat <- read_csv(filename)
```

```
## Rows: 51 Columns: 5
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): state, abb, region
## dbl (2): population, total
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

The data is imported and stored in dat. The rest of this section defines some important concepts and provides an overview of how we write code that tells R how to find the files we want to import. Chapter 38 provides more details on this topic.

5.1.1 The filesystem

You can think of your computer's filesystem as a series of nested folders, each containing other folders and files. Data scientists refer to folders as directories. We refer to the folder that contains all other folders as the root directory. We refer to the directory in which we are currently located as the working directory. The working directory therefore changes as you move through folders: think of it as your current location.

5.1.2 Relative and full paths

The path of a file is a list of directory names that can be thought of as instructions on what folders to click on, and in what order, to find the file. If these instructions are for finding the file from the root directory we refer to it as the full path. If the instructions are for finding the file starting in the working directory we refer to it as a relative path. Section 38.3 provides more details on this topic.

To see an example of a full path on your system type the following:

```
system.file(package="dslabs")
```

```
## [1] "C:/Users/user/Documents/R/win-library/4.1/dslabs"
```

The strings separated by slashes are the directory names. The first slash represents the root directory and we know this is a full path because it starts with a slash. If the first directory name appears without a slash in front, then the path is assumed to be relative. We can use the function list.files to see examples of relative paths.

```
dir <- system.file(package="dslabs")
list.files(path=dir)
```

```
##  [1] "data"        "DESCRIPTION" "extdata"     "help"        "html"
##  [6] "INDEX"       "MD5"         "Meta"        "NAMESPACE"   "R"
## [11] "script"
```

These relative paths give us the location of the files or directories if we start in the directory with the full path. For example, the full path to the help directory in the example above is /Library/Frameworks/R.framework/Versions/3.5/Resources/library/dslabs/help.

Note: You will probably not make much use of the system.file function in your day-to-day data analysis work. We introduce it in this section because it facilitates the sharing of spreadsheets by including them in the dslabs package. You will rarely have the luxury of data being included in packages you already have installed. However, you will frequently need to navigate full and relative paths and import spreadsheet formatted data.

5.1.3 The working directory

We highly recommend only writing relative paths in your code. The reason is that full paths are unique to your computer and you want your code to be portable. You can get the full path of your working directory without writing out explicitly by using the getwd function.

```
wd <- getwd()
wd
```

```
## [1] "C:/Users/user/OneDrive/Desktop/lecture/2021-2/biostatistics/github/bsms222_140_jeon"
```

If you need to change your working directory, you can use the function setwd or you can change it through RStudio by clicking on “Session.”

5.1.4 Generating path names

Another example of obtaining a full path without writing out explicitly was given above when we created the object fullpath like this:

```
filename <- "murders.csv"
dir <- system.file("extdata", package="dslabs")
fullpath <- file.path(dir, filename)
fullpath
```

```
## [1] "C:/Users/user/Documents/R/win-library/4.1/dslabs/extdata/murders.csv"
```

The function system.file provides the full path of the folder containing all the files and directories relevant to the package specified by the package argument. By exploring the directories in dir we find that the extdata contains the file we want:

```
dir <- system.file(package="dslabs")
filename %in% list.files(file.path(dir, "extdata"))
```

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

The system.file function permits us to provide a subdirectory as a first argument, so we can obtain the fullpath of the extdata directory like this:

```
dir <- system.file("extdata", package="dslabs")
dir
```

```
## [1] "C:/Users/user/Documents/R/win-library/4.1/dslabs/extdata"
```

The function file.path is used to combine directory names to produce the full path of the file we want to import.

```
fullpath <- file.path(dir, filename)
fullpath
```

```
## [1] "C:/Users/user/Documents/R/win-library/4.1/dslabs/extdata/murders.csv"
```

5.1.5 Copying files using paths

The final line of code we used to copy the file into our home directory used the function file.copy. This function takes two arguments: the file to copy and the name to give it in the new directory.

```
file.copy(fullpath, "murders_(1).csv")
```

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

If a file is copied successfully, the file.copy function returns TRUE. Note that we are giving the file the same name, murders.csv, but we could have named it anything. Also note that by not starting the string with a slash, R assumes this is a relative path and copies the file to the working directory.

You should be able to see the file in your working directory and can check by using:

```
list.files()
```

```
##  [1] "0915homework.R"
##  [2] "bsms222_140_jeon"
##  [3] "bsms222_140_jeon.Rproj"
##  [4] "Chapter_2_R_basics.Rmd"
##  [5] "Chapter2.html"
##  [6] "Chapter2.Rmd"
##  [7] "Chapter2.tex"
##  [8] "Chapter2review.Rmd"
##  [9] "Chapter3.html"
## [10] "Chapter3.Rmd"
## [11] "Chapter4_1to6.html"
## [12] "Chapter4_1to6.Rmd"
## [13] "Chapter4_from7.nb.html"
## [14] "Chapter4_from7.pdf"
## [15] "Chapter4_from7.Rmd"
## [16] "Chapter5.nb.html"
## [17] "Chapter5.pdf"
## [18] "Chapter5.Rmd"
## [19] "murders.csv"
## [20] "murders_(1).csv"
## [21] "README.md"
## [22] "SCN2A mutations in neurodevelopmental disorders.nb.html"
## [23] "SCN2A mutations in neurodevelopmental disorders.Rmd"
## [24] "test.Rmd"
```

5.2 The readr and readxl packages

In this section we introduce the main tidyverse data importing functions. We will use the murders.csv file provided by the dslabs package as an example. To simplify the illustration we will copy the file to our working directory using the following code:

```
filename <- "murders.csv"
dir <- system.file("extdata", package="dslabs")
fullpath <- file.path(dir, filename)
file.copy(fullpath, "murders_(2).csv")
```

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

5.2.1 readr

The readr library includes functions for reading data stored in text file spreadsheets into R. readr is part of the tidyverse package, or you can load it directly:

```
library(readr)
```

The following functions are available to read-in spreadsheets:

Although the suffix usually tells us what type of file it is, there is no guarantee that these always match. We can open the file to take a look or use the function read_lines to look at a few 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"
```

This also shows that there is a header. Now we are ready to read-in the data into R. From the .csv suffix and the peek at the file, we know to use read_csv:

```
dat <- read_csv(filename)
```

```
## Rows: 51 Columns: 5

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): state, abb, region
## dbl (2): population, total

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
spec(dat)
```

```
## cols(
##   state = col_character(),
##   abb = col_character(),
##   region = col_character(),
##   population = col_double(),
##   total = col_double()
## )
```

Note that we receive a message letting us know what data types were used for each column. Also note that dat is a tibble, not just a data frame. This is because read_csv is a tidyverse parser. We can confirm that the data has in fact been read-in with:

```
View(dat)
```

Finally, note that we can also use the full path for the file:

```
dat <- read_csv(fullpath)
```

```
## Rows: 51 Columns: 5

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): state, abb, region
## dbl (2): population, total

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
spec(dat)
```

```
## cols(
##   state = col_character(),
##   abb = col_character(),
##   region = col_character(),
##   population = col_double(),
##   total = col_double()
## )
```

5.2.2 readxl

You can load the readxl package using

```
library(readxl)
```

The package provides functions to read-in Microsoft Excel formats:

The Microsoft Excel formats permit you to have more than one spreadsheet in one file. These are referred to as sheets. The functions listed above read the first sheet by default, but we can also read the others. The excel_sheets function gives us the names of all the sheets in an Excel file. These names can then be passed to the sheet argument in the three functions above to read sheets other than the first.

5.3 Exercises

1. Use the read_csv function to read each of the files that the following code saves in the files object:

```
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"
```

```
read_xls(file.path(path, files[1]))
```

```
## # A tibble: 104 x 6
##    label     `INTEGRATED ALGEBRA` `GLOBAL HISTORY` `LIVING ENVIRONMENT` ENGLISH
##    <chr>                    <dbl>            <dbl>                <dbl>   <dbl>
##  1 test_year                 2010             2010                 2010    2010
##  2 Scores                  131024           113804               104201  103886
##  3 0                           56               55                   66     165
##  4 1                           NA                8                    3      69
##  5 2                            1                9                   2     237
##  6 3                           NA                3                   1     190
##  7 4                            3               15                   1     109
##  8 5                            2               11                  10     122
##  9 6                            4               29                   3     151
## 10 7                            1               37                   2     175
## # ... with 94 more rows, and 1 more variable: U.S. HISTORY <dbl>
```

```
read_csv(file.path(path, files[2]))
```

```
## Rows: 264 Columns: 2
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (2): Year, Total carbon emissions from fossil fuel consumption and cemen...
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 264 x 2
##     Year `Total carbon emissions from fossil fuel consumption and cement produc~
##    <dbl>                                                                   <dbl>
##  1  1751                                                                       3
##  2  1752                                                                       3
##  3  1753                                                                       3
##  4  1754                                                                       3
##  5  1755                                                                       3
##  6  1756                                                                       3
##  7  1757                                                                       3
##  8  1758                                                                       3
##  9  1759                                                                       3
## 10  1760                                                                       3
## # ... with 254 more rows
```

```
read_csv(file.path(path, files[3]))
```

```
## Rows: 2 Columns: 57
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (1): country
## dbl (56): 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, ...
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 2 x 57
##   country  `1960` `1961` `1962` `1963` `1964` `1965` `1966` `1967` `1968` `1969`
##   <chr>     <dbl>  <dbl>  <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   2.28   2.17
## 2 South K~   6.16   5.99   5.79   5.57   5.36   5.16   4.99   4.85   4.73   4.62
## # ... with 46 more variables: 1970 <dbl>, 1971 <dbl>, 1972 <dbl>, 1973 <dbl>,
## #   1974 <dbl>, 1975 <dbl>, 1976 <dbl>, 1977 <dbl>, 1978 <dbl>, 1979 <dbl>,
## #   1980 <dbl>, 1981 <dbl>, 1982 <dbl>, 1983 <dbl>, 1984 <dbl>, 1985 <dbl>,
## #   1986 <dbl>, 1987 <dbl>, 1988 <dbl>, 1989 <dbl>, 1990 <dbl>, 1991 <dbl>,
## #   1992 <dbl>, 1993 <dbl>, 1994 <dbl>, 1995 <dbl>, 1996 <dbl>, 1997 <dbl>,
## #   1998 <dbl>, 1999 <dbl>, 2000 <dbl>, 2001 <dbl>, 2002 <dbl>, 2003 <dbl>,
## #   2004 <dbl>, 2005 <dbl>, 2006 <dbl>, 2007 <dbl>, 2008 <dbl>, 2009 <dbl>, ...
```

```
read_csv(file.path(path, files[4]))
```

```
## Rows: 95 Columns: 1
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): Sun 4.8 5840 G2
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 95 x 1
##    `Sun 4.8 5840 G2`
##    <chr>
##  1 SiriusA 1.4 9620 A1
##  2 Canopus -3.1 7400 F0
##  3 Arcturus -0.4 4590 K2
##  4 AlphaCentauriA 4.3 5840 G2
##  5 Vega 0.5 9900 A0
##  6 Capella -0.6 5150 G8
##  7 Rigel -7.2 12140 B8
##  8 ProcyonA 2.6 6580 F5
##  9 Betelgeuse -5.7 3200 M2
## 10 Achemar -2.4 20500 B3
## # ... with 85 more rows
```

```
read_csv(file.path(path, files[5]))
```

```
## Rows: 2 Columns: 113
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr   (1): country
## dbl (112): 1960_fertility, 1960_life_expectancy, 1961_fertility, 1961_life_e...
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## # A tibble: 2 x 113
##   country     `1960_fertility` `1960_life_expe~ `1961_fertility` `1961_life_expe~
##   <chr>                  <dbl>            <dbl>            <dbl>            <dbl>
## 1 Germany                 2.41             69.3             2.44             69.8
## 2 South Korea             6.16             53.0             5.99             53.8
## # ... with 108 more variables: 1962_fertility <dbl>,
## #   1962_life_expectancy <dbl>, 1963_fertility <dbl>,
## #   1963_life_expectancy <dbl>, 1964_fertility <dbl>,
## #   1964_life_expectancy <dbl>, 1965_fertility <dbl>,
## #   1965_life_expectancy <dbl>, 1966_fertility <dbl>,
## #   1966_life_expectancy <dbl>, 1967_fertility <dbl>,
## #   1967_life_expectancy <dbl>, 1968_fertility <dbl>, ...
```

```
read_csv(file.path(path, files[6]))
```

```
## Rows: 51 Columns: 5

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): state, abb, region
## dbl (2): population, total

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # 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
```

```
read_csv(file.path(path, files[7]))
```

```
## New names:
## * `` -> ...1

## Rows: 572 Columns: 11

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): Region, eicosenoic
## dbl (9): ...1, Area, palmitic, palmitoleic, stearic, oleic, linoleic, linole...
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## Warning: One or more parsing issues, see `problems()` for details
```

```
## # A tibble: 572 x 11
##     ...1 Region        Area palmitic palmitoleic stearic oleic linoleic linolenic
##    <dbl> <chr>        <dbl>    <dbl>       <dbl>   <dbl> <dbl>    <dbl>     <dbl>
##  1     1 North-Apulia     1        1        1075      75   226     7823       672
##  2     2 North-Apulia     1        1        1088      73   224     7709       781
##  3     3 North-Apulia     1        1         911      54   246     8113       549
##  4     4 North-Apulia     1        1         966      57   240     7952       619
##  5     5 North-Apulia     1        1        1051      67   259     7771       672
##  6     6 North-Apulia     1        1         911      49   268     7924       678
##  7     7 North-Apulia     1        1         922      66   264     7990       618
##  8     8 North-Apulia     1        1        1100      61   235     7728       734
##  9     9 North-Apulia     1        1        1082      60   239     7745       709
## 10    10 North-Apulia     1        1        1037      55   213     7944       633
## # ... with 562 more rows, and 2 more variables: arachidic <dbl>,
## #   eicosenoic <chr>
```

```r
read_csv(file.path(path, files[8]))
```

```
## Rows: 14244 Columns: 1
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): %PDF-1.5
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## Warning: One or more parsing issues, see `problems()` for details
```

```
## # A tibble: 14,244 x 1
##    `%PDF-1.5`
##    <chr>
##  1 "%\xb5\xb5\xb5\xb5"
##  2 "1 0 obj"
##  3 "<</Type/Catalog/Pages 2 0 R/Lang(es-ES) /StructTreeRoot 32 0 R/MarkInfo<</M~
##  4 "endobj"
##  5 "2 0 obj"
##  6 "<</Type/Pages/Count 12/Kids[ 4 0 R 10 0 R 12 0 R 14 0 R 16 0 R 18 0 R 20 0 ~
##  7 "endobj"
##  8 "3 0 obj"
##  9 "<</Author(Maria M. Juiz Gallego) /CreationDate(D:20180604163453-04'00') /Mo~
## 10 "endobj"
## # ... with 14,234 more rows
```

```
read_csv(file.path(path, files[9]))
```

```
## Rows: 240 Columns: 5

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): Sex
## dbl (3): Age, DeathProb, LifeExp

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 240 x 5
##      Age Sex   DeathProb NumberOfLives LifeExp
##    <dbl> <chr>     <dbl>         <dbl>   <dbl>
##  1     0 Male   0.00638         100000    76.2
##  2     1 Male   0.000453         99362    75.6
##  3     2 Male   0.000282         99317    74.7
##  4     3 Male   0.00023          99289    73.7
##  5     4 Male   0.000169         99266    72.7
##  6     5 Male   0.000155         99249    71.7
##  7     6 Male   0.000145         99234    70.7
##  8     7 Male   0.000135         99219    69.7
##  9     8 Male   0.00012          99206    68.8
## 10     9 Male   0.000105         99194    67.8
## # ... with 230 more rows
```

2. Note that the last one, the olive file, gives us a warning. This is because the first line of the file is missing the header for the first column.

Read the help file for read_csv to figure out how to read in the file without reading this header. If you skip the header, you should not get this warning. Save the result to an object called dat.

```
# Warning: One or more parsing issues, see `problems()` for details
?read_csv
```

```
## starting httpd help server ... done
```

```
dat <- read_csv(file.path(path, files[7]), skip=1)
```

```
## New names:
## * `1` -> `1...1`
## * `1` -> `1...3`
## * `1` -> `1...4`
## Rows: 571 Columns: 12
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (1): North-Apulia
## dbl (11): 1...1, 1...3, 1...4, 1075, 75, 226, 7823, 672, 36, 60, 29
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
dat
```

```
## # A tibble: 571 x 12
##    `1...1` `North-Apulia` `1...3` `1...4` `1075`  `75` `226` `7823` `672`  `36`
##      <dbl> <chr>            <dbl>   <dbl>  <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>
##  1       2 North-Apulia         1       1   1088    73   224   7709   781    31
##  2       3 North-Apulia         1       1    911    54   246   8113   549    31
##  3       4 North-Apulia         1       1    966    57   240   7952   619    50
##  4       5 North-Apulia         1       1   1051    67   259   7771   672    50
##  5       6 North-Apulia         1       1    911    49   268   7924   678    51
##  6       7 North-Apulia         1       1    922    66   264   7990   618    49
##  7       8 North-Apulia         1       1   1100    61   235   7728   734    39
##  8       9 North-Apulia         1       1   1082    60   239   7745   709    46
##  9      10 North-Apulia         1       1   1037    55   213   7944   633    26
## 10      11 North-Apulia         1       1   1051    35   219   7978   605    21
## # ... with 561 more rows, and 2 more variables: 60 <dbl>, 29 <dbl>
```

3. A problem with the previous approach is that we don’t know what the columns represent. Type:

```
names(dat)
```

```
##  [1] "1...1"          "North-Apulia" "1...3"          "1...4"          "1075"
##  [6] "75"             "226"          "7823"           "672"            "36"
## [11] "60"             "29"
```

to see that the names are not informative.

Use the readLines function to read in just the first line (we later learn how to extract values from the output).

```
readLines(file.path(path, files[7]), n=1)
```

```
## [1] ",Region,Area,palmitic,palmitoleic,stearic,oleic,linoleic,linolenic,arachidic,eicosenoic"
```

5.4 Downloading files

Another common place for data to reside is on the internet. When these data are in files, we can download them and then import them or even read them directly from the web. For example, we note that because our dslabs package is on GitHub, the file we downloaded with the package has a url:

```
url <- "https://raw.githubusercontent.com/rafalab/dslabs/master/inst/extdata/murders.csv"
```

The read_csv file can read these files directly:

```
dat <- read_csv(url)
```

```
## Rows: 51 Columns: 5
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): state, abb, region
## dbl (2): population, total
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
dat
```

```
## # 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
```

If you want to have a local copy of the file, you can use the download.file function:

```
download.file(url, "murders_(3).csv")
```

This will download the file and save it on your system with the name murders.csv. You can use any name here, not necessarily murders.csv. Note that when using download.file you should be careful as it will overwrite existing files without warning.

Two functions that are sometimes useful when downloading data from the internet are tempdir and tempfile. The first creates a directory with a random name that is very likely to be unique. Similarly, tempfile creates a character string, not a file, that is likely to be a unique filename. So you can run a command like this which erases the temporary file once it imports the data:

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

```
## Rows: 51 Columns: 5

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): state, abb, region
## dbl (2): population, total

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

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

```
## Warning in file.remove(tmp_filename):   'C:
## \Users\user\AppData\Local\Temp\RtmpGOcK6V\file27b85d744431'        ,
##     'Permission denied'

## [1] FALSE
```

5.5 R-base importing functions

R-base also provides import functions. These have similar names to those in the tidyverse, for example read.table, read.csv and read.delim. You can obtain an data frame like dat using:

```
dat2 <- read.csv(filename)
dat2
```

```
##                   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
## 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
## 11              Georgia  GA         South    9920000   376
## 12               Hawaii  HI          West    1360301     7
## 13                Idaho  ID          West    1567582    12
## 14             Illinois  IL North Central   12830632   364
## 15              Indiana  IN North Central    6483802   142
## 16                 Iowa  IA North Central    3046355    21
## 17               Kansas  KS North Central    2853118    63
## 18             Kentucky  KY         South    4339367   116
## 19            Louisiana  LA         South    4533372   351
## 20                Maine  ME     Northeast    1328361    11
## 21             Maryland  MD         South    5773552   293
## 22        Massachusetts  MA     Northeast    6547629   118
## 23             Michigan  MI North Central    9883640   413
## 24            Minnesota  MN North Central    5303925    53
## 25          Mississippi  MS         South    2967297   120
## 26             Missouri  MO North Central    5988927   321
## 27              Montana  MT          West     989415    12
## 28             Nebraska  NE North Central    1826341    32
## 29               Nevada  NV          West    2700551    84
## 30        New Hampshire  NH     Northeast    1316470     5
## 31           New Jersey  NJ     Northeast    8791894   246
## 32           New Mexico  NM          West    2059179    67
## 33             New York  NY     Northeast   19378102   517
## 34       North Carolina  NC         South    9535483   286
## 35         North Dakota  ND North Central     672591     4
## 36                 Ohio  OH North Central   11536504   310
## 37             Oklahoma  OK         South    3751351   111
## 38               Oregon  OR          West    3831074    36
## 39         Pennsylvania  PA     Northeast   12702379   457
## 40         Rhode Island  RI     Northeast    1052567    16
## 41       South Carolina  SC         South    4625364   207
## 42         South Dakota  SD North Central     814180     8
## 43            Tennessee  TN         South    6346105   219
## 44                Texas  TX         South   25145561   805
## 45                 Utah  UT          West    2763885    22
## 46              Vermont  VT     Northeast    625741     2
## 47             Virginia  VA         South   8001024   250
## 48           Washington  WA          West   6724540    93
## 49        West Virginia  WV         South   1852994    27
## 50            Wisconsin  WI North Central   5686986    97
## 51              Wyoming  WY          West    563626     5
```

An often useful R-base importing function is scan, as it provides much flexibility. When reading in spreadsheets many things can go wrong. The file might have a multiline header, be missing cells, or it might use an unexpected encoding. We recommend you read this post about common issues found here: https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely-positively-must-know-about-unicode-and-character-sets-no-excuses/.

With experience you will learn how to deal with different challenges. Carefully reading the help files for the functions discussed here will be useful. With scan you can read-in each cell of a file. Here is an example:

```
path <- system.file("extdata", package="dslabs")
filename <- "murders.csv"
x <- scan(file.path(path, filename), sep=",", what="c")
x[1:10]
```

```
##  [1] "state"      "abb"        "region"     "population" "total"
##  [6] "Alabama"    "AL"         "South"      "4779736"    "135"
```

Note that the tidyverse provides read_lines, a similarly useful function.

5.6 Text versus binary files

For data science purposes, files can generally be classified into two categories: text files (also known as ASCII files) and binary files. You have already worked with text files. All your R scripts are text files and so are the R markdown files used to create this book. The csv tables you have read are also text files. One big advantage of these files is that we can easily "look" at them without having to purchase any kind of special software or follow complicated instructions. Any text editor can be used to examine a text file, including freely available editors such as RStudio, Notepad, textEdit, vi, emacs, nano, and pico. To see this, try opening a csv file using the "Open file" RStudio tool. You should be able to see the content right on your editor. However, if you try to open, say, an Excel xls file, jpg or png file, you will not be able to see anything immediately useful. These are binary files. Excel files are actually compressed folders with several text files inside. But the main distinction here is that text files can be easily examined.

Although R includes tools for reading widely used binary files, such as xls files, in general you will want to find data sets stored in text files. Similarly, when sharing data you want to make it available as text files as long as storage is not an issue (binary files are much more efficient at saving space on your disk). In general, plain-text formats make it easier to share data since commercial software is not required for working with the data.

Extracting data from a spreadsheet stored as a text file is perhaps the easiest way to bring data from a file to an R session. Unfortunately, spreadsheets are not always available and the fact that you can look at text files does not necessarily imply that extracting data from them will be straightforward. In the Data Wrangling part of the book we learn to extract data from more complex text files such as html files.

5.7 Unicode versus ASCII

A pitfall in data science is assuming a file is an ASCII text file when, in fact, it is something else that can look a lot like an ASCII text file: a Unicode text file.

To understand the difference between these, remember that everything on a computer needs to eventually be converted to 0s and 1s. ASCII is an encoding that maps characters to numbers. ASCII uses 7 bits (0s and 1s) which results in $2^7 = 128$ unique items, enough to encode all the characters on an English language keyboard. However, other languages use characters not included in this encoding. For example, the é in México is not encoded by ASCII. For this reason, a new encoding, using more than 7 bits, was defined: Unicode. When using Unicode, one can chose between 8, 16, and 32 bits abbreviated UTF-8, UTF-16, and UTF-32 respectively. RStudio actually defaults to UTF-8 encoding.

Although we do not go into the details of how to deal with the different encodings here, it is important that you know these different encodings exist so that you can better diagnose a problem if you encounter it. One way problems manifest themselves is when you see "weird looking" characters you were not expecting. This StackOverflow discussion is an example: https://stackoverflow.com/questions/18789330/r-on-windows-character-encoding-hell.

5.8 Organizing data with spreadsheets

Although this book focuses almost exclusively on data analysis, data management is also an important part of data science. As explained in the introduction, we do not cover this topic. However, quite often data analysts needs to collect data, or work with others collecting data, in a way that is most conveniently stored in a spreadsheet. Although filling out a spreadsheet by hand is a practice we highly discourage, we instead recommend the process be automatized as much as possible, sometimes you just have to do it. Therefore, in this section, we provide recommendations on how to organize data in a spreadsheet. Although there are R packages designed to read Microsoft Excel spreadsheets, we generally want to avoid this format. Instead, we recommend Google Sheets as a free software tool. Below we summarize the recommendations made in paper by Karl Broman and Kara Woo. Please read the paper for important details.

Be Consistent - Before you commence entering data, have a plan. Once you have a plan, be consistent and stick to it. Choose Good Names for Things - You want the names you pick for objects, files, and directories to be memorable, easy to spell, and descriptive. This is actually a hard balance to achieve and it does require time and thought. One important rule to follow is do not use spaces, use underscores _ or dashes instead -. Also, avoid symbols; stick to letters and numbers. Write Dates as YYYY-MM-DD - To avoid confusion, we strongly recommend using this global ISO 8601 standard. No Empty Cells - Fill in all cells and use some common code for missing data. Put Just One Thing in a Cell - It is better to add columns to store the extra information rather than having more than one piece of information in one cell. Make It a Rectangle - The spreadsheet should be a rectangle. Create a Data Dictionary - If you need to explain things, such as what the columns are or what the labels used for categorical variables are, do this in a separate file. No Calculations in the Raw Data Files - Excel permits you to perform calculations. Do not make this part of your spreadsheet. Code for calculations should be in a script. Do Not Use Font Color or Highlighting as Data - Most import functions are not able to import this information. Encode this information as a variable instead. Make Backups - Make regular backups of your data. Use Data Validation to Avoid Errors - Leverage the tools in your spreadsheet software so that the process is as error-free and repetitive-stress-injury-free as possible. Save the Data as Text Files - Save files for sharing in comma or tab delimited format.

5.9 Exercises

1. Pick a measurement you can take on a regular basis. For example, your daily weight or how long it takes you to run 5 miles. Keep a spreadsheet that includes the date, the hour, the measurement, and any other informative variable you think is worth keeping. Do this for 2 weeks. Then make a plot.

```
# Plan: Weather for Dongdaemun-go, Seoul
# date, time(00,06, 12, 18), temperature, humidity, PM, etc.
# 210924~
```