Chapter 3

Getting Data into R

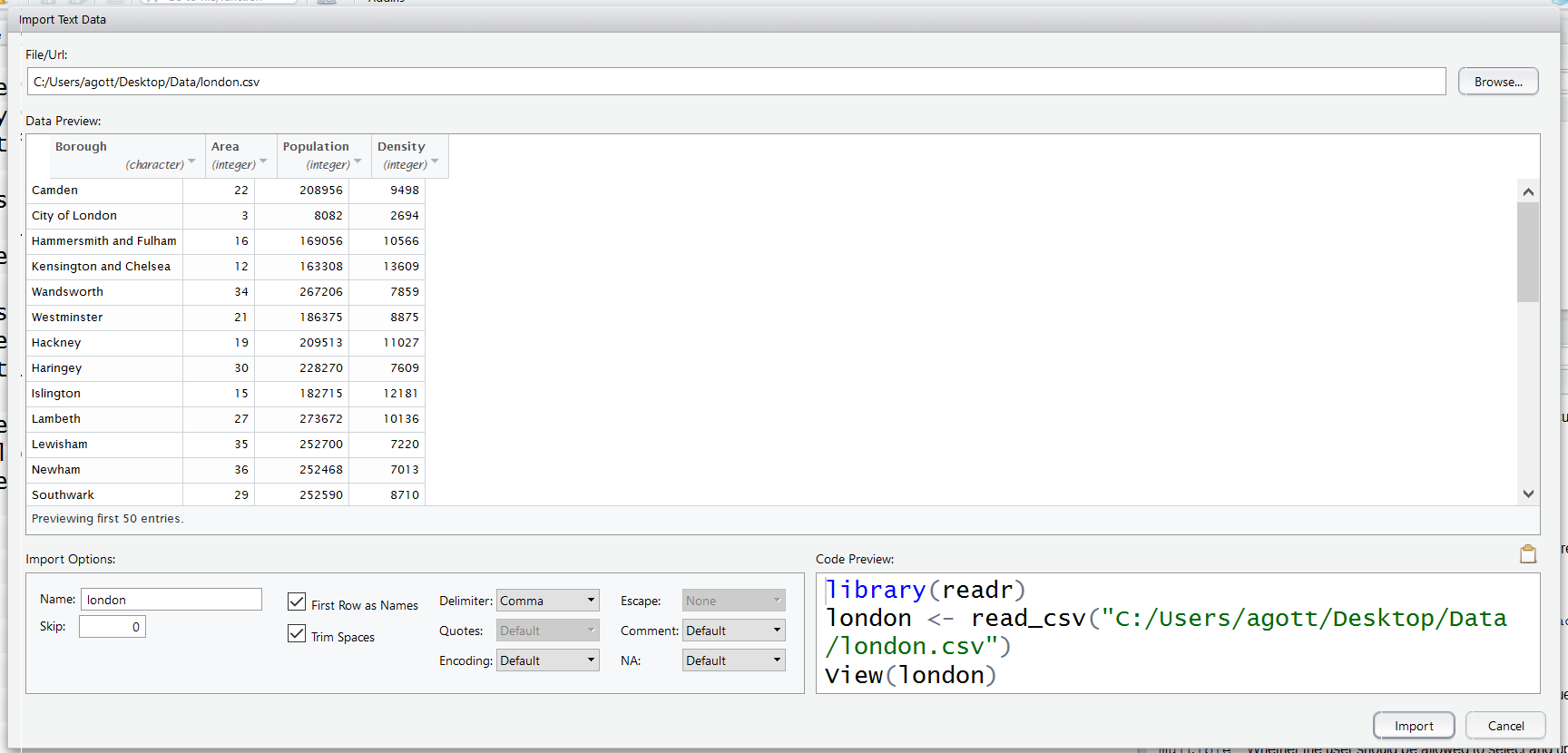
## Importing Data

The **readr** package provides lots of convenient functions for reading in various files of different formats. Alternatively, data can also be imported using the RStudio interface and code is generated for you.

|  |
| --- |
| > library(readr) |

### Tabular Data

We can load data using RStudio via File > Import Dataset > From Text (readr). This will open the file browser window where you can select the file you wish to import. Once you have selected the file, the import wizard will open in which you can set the import parameters.



If you are happy with the parameter values, you can click on Import. The R code that is then generated and run is displayed in the Console. The data that is read is immediately shown.

In the **readr** package, we use the read\_csv function to import data. This function allows us to set the data types of each column (we will return to this later), meaning that we can ensure that our data is imported correctly. If we don't supply types the function will guess and return the code that was used for specifying columns.

|  |
| --- |
| > london <- read\_csv("london.csv")  Parsed with column specification:  cols(  Borough = col\_character(),  Area = col\_integer(),  Population = col\_integer(),  Density = col\_integer()  ) |

|  |  |
| --- | --- |
|  | If we have missing data we can add an option na where we can specify how missing values are represented in the data. For example:  london <- read\_csv("london.csv", na = "NA") |

### File paths in R

To be able to work with a file in R, the location of the file must be referenced in some way. The file may be stored in a folder on your computer, on an external drive, or on a server.

The simplest way is to use the full path. In Windows, paths are specified using backslashes, but in R, backslashes already have a special meaning, so each backslash should be replaced with something else instead. This can either be two backslashes ("\\") or a forward slash ("/").

|  |
| --- |
| > # for example  > london <- read\_csv("C:/Users/agott/Documents/london.csv")  > london <- read\_csv("F:/london.csv") # if stored on F drive  > london <- read\_csv("//myserver/london.csv") # server example |

|  |  |
| --- | --- |
|  | To save typing, the character "~" is a shortcut for the Documents folder on your computer. For example,  > london <- read\_csv("C:/Users/agott/Documents/london.csv")  is the same as  > london <- read\_csv("~/london.csv") |

Alternatively, the path can be relative to your “working directory”. This is the folder that R will use to look for files. This can be set using the function setwd. To see your current working directory, use getwd.

|  |
| --- |
| > getwd()  [1] "C:/Users/agott"  > setwd("C:/Users/agott/Desktop/Data/")  > getwd()  [1] "C:/Users/agott/Desktop/Data" |

If you have set up a project in RStudio, then the working directory will be set to the location of the project.

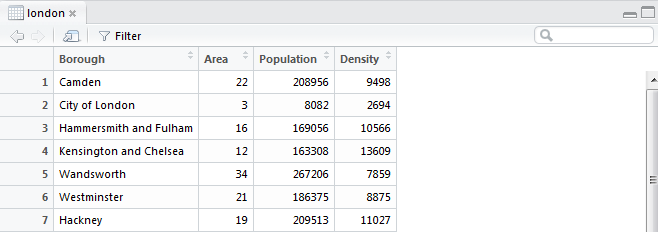
## Viewing Data

Once we have imported our data we typically want to quickly preview the data. There are a number of ways of doing this in R.

### View

The View function opens a special preview window showing the data and, in RStudio, allowing us to interact with the data.

|  |
| --- |
| > View(london) |



### Printing in the Console

When we create any data object in R we can simply enter the name of the object in the console to see that object.

|  |
| --- |
| > london  # A tibble: 33 x 4  Borough Area Population Density  <chr> <int> <int> <int>  1 Camden 22 208956 9498  2 City of London 3 8082 2694  3 Hammersmith and Fulham 16 169056 10566  4 Kensington and Chelsea 12 163308 13609  5 Wandsworth 34 267206 7859  6 Westminster 21 186375 8875  7 Hackney 19 209513 11027  8 Haringey 30 228270 7609  9 Islington 15 182715 12181  10 Lambeth 27 273672 10136  # ... with 23 more rows |

### Previewing the Start and End of the Data

R contains a number of utility functions for viewing data. In order to take a quick look at the first or last few rows of the data we can use the functions head or tail.

|  |
| --- |
| > head(london)  Borough Area Population Density  1 Camden 22 208956 9498  2 City of London 3 8082 2694  3 Hammersmith and Fulham 16 169056 10566  4 Kensington and Chelsea 12 163308 13609  5 Wandsworth 34 267206 7859  6 Westminster 21 186375 8875  > head(london, n = 3) # the first 3 rows  Borough Area Population Density  1 Camden 22 208956 9498  2 City of London 3 8082 2694  3 Hammersmith and Fulham 16 169056 10566 |

|  |  |
| --- | --- |
|  | 1. Import the “olympics.csv” data into R 2. View the data. How many rows and columns does this data have? 3. Print the last 10 rows |

## Data Frames

### Overview

A data frame (technically “data.frame”) is R’s standard tabular structure for data. It is the default structure used when importing data. There are several inbuilt example data frames contained within the datasets package. The functions in **readr** actually import data as “tibbles”. These are effectively just a special case of data frames that appear differently when printed to the console.

### Data Frame Attributes

There are lots of useful functions to extract information about the data frame’s structure.

|  |  |
| --- | --- |
| Function | Description |
| nrow | Number of rows |
| ncol | Number of columns |
| dim | Dimension of the data frame (rows, columns) |
| names | Column names |

|  |
| --- |
| > nrow(london)  [1] 33  >  > ncol(london)  [1] 4  >  > dim(london)  [1] 33 4  >  > names(london)  [1] "Borough" "Area" "Population" "Density" |

|  |  |
| --- | --- |
|  | These functions have returned new objects, which are not in a tabular shape. They are not data frames, but are in fact another type of object, called a "vector". Notice how they are printed differently. |

## Exporting Data

If you wish to export a data frame to a csv file we use the write\_csv function.

|  |
| --- |
| > write\_csv(london, "newLondon.csv") |

|  |  |
| --- | --- |
|  | 1. Export the airquality data into a csv file 2. Open the file to check that it has been exported correctly. 3. Can you change the way missing values are exported so that in the output csv file they are represented as the string "MISSING"? |

## Importing Other Data Formats

So far, we have shown how to read from and write to csv. These are universal file formats, so whatever the source of the data, it is usually possible to convert it and save it as a csv file, before reading into R. However, it is much more convenient to be able to read directly from the source itself. In this course we will also consider reading from Excel.

### Reading from Excel using readxl

There are many different packages available to read from and write to Excel files. Most of them have some external dependencies such as Java or a zip application. Examples include **xlsx**, **XLConnect**, or **openxlsx**. In this section we are going to cover **readxl**, which is a minimal R package that only reads from Excel files. It does not have any external dependencies, and the package only contains two functions:

|  |  |
| --- | --- |
| Function | Description |
| excel\_sheets | Sheet names |
| read\_excel | Given a sheet name and path, reads tabular data from the sheet in the Excel file |

|  |
| --- |
| > excel\_sheets("iris.xlsx")  [1] "iris"  > irisXL <- read\_excel("iris.xlsx", sheet = "iris")  > irisXL  Sepal.Length Sepal.Width Petal.Length Petal.Width Species  1 5.1 3.5 1.4 0.2 setosa  2 4.9 3.0 1.4 0.2 setosa  3 4.7 3.2 1.3 0.2 setosa  4 4.6 3.1 1.5 0.2 setosa  5 5.0 3.6 1.4 0.2 setosa  6 5.4 3.9 1.7 0.4 setosa  ... |

|  |  |
| --- | --- |
|  | 1. Read in the "Medals" sheet from Rio2016.xlsx into an object called medals 2. View the data 3. Print the first 6 rows of Medals 4. How many rows does the data have? 5. What are the column names? |

## Data Columns (Vectors)

### Overview

Data frames are made up of columns which we call vectors. Vectors are an extremely useful data structure in their own right; one that we will revisit throughout this course.

### Extracting Columns

Columns (vectors) may be extracted from a data frame using the “$” symbol.

|  |
| --- |
| > airquality$Wind # Extract the Wind column  [1] 7.4 8.0 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 6.9 9.7  [13] 9.2 10.9 13.2 11.5 12.0 18.4 11.5 9.7 9.7 16.6 9.7 12.0  [25] 16.6 14.9 8.0 12.0 14.9 5.7 7.4 8.6 9.7 16.1 9.2 8.6  [37] 14.3 9.7 6.9 13.8 11.5 10.9 9.2 8.0 13.8 11.5 14.9 20.7  ... |

### Creating Vectors

We often need to create our own vectors. In this course we will use vectors to:

* Demonstrate R functions
* Relabel categorical data
* Search for patterns of text in a data frame
* Set axis limits and labels
* Specify colours to use on a plot

The simplest method for creating vectors is to “combine” values using the c function.

|  |
| --- |
| > c(4, 3, 6, 5) # A simple numeric vector  [1] 4 3 6 5  > c("A", "B", "C") # A simple character vector  [1] "A" "B" "C"  > c(FALSE, TRUE, FALSE) # A simple logical vector  [1] FALSE TRUE FALSE |

There are also a number of utility functions that help us to create vectors. Some examples of these creating vectors are provided below.

|  |
| --- |
| > # Integer sequence  > 1:10  [1] 1 2 3 4 5 6 7 8 9 10  >  > # More complicated sequence  > seq(from = 0, to = 5, by = 0.5)  [1] 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0  >  > # Repeated values  > rep("Analytics", 3)  [1] "Analytics" "Analytics" "Analytics" |

### Subscripting Vectors

Once we have created a vector, we need to be able to easily extract required information. We “subscript” a vector using square brackets as follows:

Vector [ Elements to return from the vector ]

The square brackets can contain one of 5 possible types of information, depending on the elements we wish to return:

|  |  |
| --- | --- |
| Subscript Type | Description |
| Blank | Return all elements (no subscripting performed) |
| Vector of positive integers | Indices of elements to return |
| Vector of negative integers | Indices of elements to omit |
| Vector of logical values | Only “TRUE” values are returned |
| Vector of characters | Names of elements to return |

Although vector subscripting is not the primary focus of this course, positive integer subscripting is sometimes used to generate example data.

|  |
| --- |
| > LETTERS[1:5]  [1] "A" "B" "C" "D" "E"  > LETTERS[c(1,3,5)]  [1] "A" "C" "E" |