Reading and Writing CSV Files

The most common way that scientists store data is in Excel spreadsheets. While there are R packages designed to access data from Excel spreadsheets (e.g., gdata, RODBC, XLConnect, xlsx, RExcel), users often find it easier to save their spreadsheets in [comma-separated values](%7B%7B%20page.root%20%7D%7D/reference.html#comma-separated-values-csv) files (CSV) and then use R’s built in functionality to read and manipulate the data. In this short lesson, we’ll learn how to read data from a .csv and write to a new .csv, and explore the [arguments](%7B%7B%20page.root%20%7D%7D/reference.html#argument) that allow you read and write the data correctly for your needs.

### Read a .csv and Explore the Arguments

Let’s start by opening a .csv file containing information on the speeds at which cars of different colors were clocked in 45 mph zones in the four-corners states (CarSpeeds.csv). We will use the built in read.csv(...) [function call](%7B%7B%20page.root%20%7D%7D/reference.html#function-call), which reads the data in as a data frame, and assign the data frame to a variable (using <-) so that it is stored in R’s memory. Then we will explore some of the basic arguments that can be supplied to the function.

# First, set your working directory (see episode 'Analyzing Patient Data' for more  
# info)  
setwd("~/Desktop/r-novice-inflammation/")

{: .language-r}

# Import the data and look at the first six rows  
carSpeeds <- read.csv(file = 'data/car-speeds.csv')  
head(carSpeeds)

{: .language-r}

Color Speed State  
1 Blue 32 NewMexico  
2 Red 45 Arizona  
3 Blue 35 Colorado  
4 White 34 Arizona  
5 Red 25 Arizona  
6 Blue 41 Arizona

{: .output}

## Changing Delimiters

The default delimiter of the read.csv() function is a comma, but you can use other delimiters by supplying the ‘sep’ argument to the function (e.g., typing sep = ';' allows a semi-colon separated file to be correctly imported - see ?read.csv() for more information on this and other options for working with different file types). {: .callout}

The call above will import the data, but we have not taken advantage of several handy arguments that can be helpful in loading the data in the format we want. Let’s explore some of these arguments.

### The header Argument

The default for read.csv(...) is to set the header argument to TRUE. This means that the first row of values in the .csv is set as header information (column names). If your data set does not have a header, set the header argument to FALSE:

# The first row of the data without setting the header argument:  
carSpeeds[1, ]

{: .language-r}

Color Speed State  
1 Blue 32 NewMexico

{: .output}

# The first row of the data if the header argument is set to FALSE:  
carSpeeds <- read.csv(file = 'data/car-speeds.csv', header = FALSE)  
  
carSpeeds[1, ]

{: .language-r}

V1 V2 V3  
1 Color Speed State

{: .output}

Clearly this is not the desired behavior for this data set, but it may be useful if you have a dataset without headers.

### The stringsAsFactors Argument

When a file is imported with read.csv(), R determines the appropriate data type for each column. A column containing integer values will be imported as typ int; a column containing character string values (e.g. names) will be imported as type chr. We can confirm the import type of each column in a data frame with the str() function, which returns information about the structure of an R object. (The str() function will be reviewed a little more in the lesson [Data Types and Structures](%7B%7B%20page.root%20%7D%7D/13-supp-data-structures/.)

# Import the data file  
carSpeeds <- read.csv(file = 'data/car-speeds.csv', header = TRUE)  
  
# Display the data type and first few values of each column with str()  
str(carSpeeds)

{: .language-r}

'data.frame': 100 obs. of 3 variables:  
 $ Color: chr "Blue" " Red" "Blue" "White" ...  
 $ Speed: int 32 45 35 34 25 41 34 29 31 26 ...  
 $ State: chr "NewMexico" "Arizona" "Colorado" "Arizona" ...

{: .output}

For string data measures that can hold any value (for example, first names) it is appropriate to treat the data as character strings. However, string data columns are often used to represent categorical data measures, such as gender, or political party, which are restricted to a small set of unique values. In R, a categorical data type is called a **factor** and the set of unique values it contains are called **levels**. Correctly treating categorical data as a factor protects us against data entry errors, because R will not allow us to enter a value into a factor data column that does not match the defined set of levels.

To instruct R to import string data columns as factors, we set the stringsAsFactors argument to read.csv() to TRUE.

# Set stringsAsFactors = TRUE  
carSpeeds <- read.csv(file = 'data/car-speeds.csv', header = TRUE, stringsAsFactors = TRUE)  
  
# The Color and State columns are now treated as factors  
str(carSpeeds)

{: .language-r}

'data.frame': 100 obs. of 3 variables:  
 $ Color: Factor w/ 5 levels " Red","Black",..: 3 1 3 5 4 3 3 2 5 4 ...  
 $ Speed: int 32 45 35 34 25 41 34 29 31 26 ...  
 $ State: Factor w/ 4 levels "Arizona","Colorado",..: 3 1 2 1 1 1 3 2 1 2 ...

{: .output}

If we make a data entry error when trying to add a new car to our data frame, R will catch it.

# Assume we spell a state name incorrectly...  
newCarData <- c("Blue", "22", "Otah")  
  
# We try to add it to the data frame with the rbind command  
carSpeeds <- rbind(carSpeeds, newCarData)

{: .language-r}

Warning in `[<-.factor`(`\*tmp\*`, ri, value = "Otah"): invalid factor level, NA  
generated

{: .warning}

R has warned us that we have supplied an invalid factor level, and that it has generated “NA” (not available; missing data).

# See the row we added by printing the tail (last six rows) of the data frame  
tail(carSpeeds)

{: .language-r}

Color Speed State  
96 Black 46 Colorado  
97 White 34 Utah  
98 Black 25 NewMexico  
99 Black 32 Arizona  
100 White 42 Utah  
101 Blue 22 <NA>

{: .output}

In addition to protecting us from data entry errors, using factors can insure that we perform appropriate categorical data analyses. For example, compare the behaviour of the summary() function when our data columns are treated as character strings or factors.

# Import string data as chr (no stringsAsFactors argument)  
carSpeeds <- read.csv(file = 'data/car-speeds.csv')  
summary(carSpeeds)

{: .language-r}

Color Speed State   
 Length:100 Min. :25.00 Length:100   
 Class :character 1st Qu.:32.00 Class :character   
 Mode :character Median :38.50 Mode :character   
 Mean :38.71   
 3rd Qu.:45.25   
 Max. :54.00

{: .output}

# Import string data as factors (stringsAsFactors = TRUE)  
carSpeeds <- read.csv(file = 'data/car-speeds.csv', stringsAsFactors = TRUE)  
summary(carSpeeds)

{: .language-r}

Color Speed State   
 Red : 1 Min. :25.00 Arizona :23   
 Black:21 1st Qu.:32.00 Colorado :28   
 Blue :28 Median :38.50 NewMexico:20   
 Red :28 Mean :38.71 Utah :29   
 White:22 3rd Qu.:45.25   
 Max. :54.00

{: .output}

When our string data are imported as chr, summary() tells us only the number of rows and the data type. However, when our string data are imported as factors, summary() provides the more useful frequency table, showing the number of occurrences of each factor level in the column.

However, when working with factors, it is more difficult to perform certain operations, such as replacing values. For example, let’s say we find out that the data collector had some red/green colour blindness, and had accidentally recorded green cars as being blue. In order to correct the data set, we can replace ‘Blue’ with ‘Green’ in the $Color column:

# Import string data as factors  
carSpeeds <- read.csv(file = 'data/car-speeds.csv', stringsAsFactors = TRUE)  
  
# Here we will use R's `ifelse` function, in which we provide the test phrase,  
# the outcome if the result of the test is 'TRUE', and the outcome if the  
# result is 'FALSE'. We will also assign the results to the Color column,  
# using '<-'  
  
carSpeeds$Color <- ifelse(carSpeeds$Color == 'Blue', 'Green', carSpeeds$Color)  
carSpeeds$Color

{: .language-r}

[1] "Green" "1" "Green" "5" "4" "Green" "Green" "2" "5"   
 [10] "4" "4" "5" "Green" "Green" "2" "4" "Green" "Green"  
 [19] "5" "Green" "Green" "Green" "4" "Green" "4" "4" "4"   
 [28] "4" "5" "Green" "4" "5" "2" "4" "2" "2"   
 [37] "Green" "4" "2" "4" "2" "2" "4" "4" "5"   
 [46] "2" "Green" "4" "4" "2" "2" "4" "5" "4"   
 [55] "Green" "Green" "2" "Green" "5" "2" "4" "Green" "Green"  
 [64] "5" "2" "4" "4" "2" "Green" "5" "Green" "4"   
 [73] "5" "5" "Green" "Green" "Green" "Green" "Green" "5" "2"   
 [82] "Green" "5" "2" "2" "4" "4" "5" "5" "5"   
 [91] "5" "4" "4" "4" "5" "2" "5" "2" "2"   
[100] "5"

{: .output}

What happened?!? It looks like ‘Blue’ was replaced with ‘Green’, but every other color was turned into a number (as a character string, given the quote marks before and after). This is because the colors of the cars were loaded as factors, and the factor level was reported following replacement.

If we load the dataset without setting stringsAsFactors to TRUE, the operation to replace ‘Blue’ with ‘Green’ in the $Color column works as expected:

carSpeeds <- read.csv(file = 'data/car-speeds.csv')  
str(carSpeeds)

{: .language-r}

'data.frame': 100 obs. of 3 variables:  
 $ Color: chr "Blue" " Red" "Blue" "White" ...  
 $ Speed: int 32 45 35 34 25 41 34 29 31 26 ...  
 $ State: chr "NewMexico" "Arizona" "Colorado" "Arizona" ...

{: .output}

carSpeeds$Color <- ifelse(carSpeeds$Color == 'Blue', 'Green', carSpeeds$Color)  
carSpeeds$Color

{: .language-r}

[1] "Green" " Red" "Green" "White" "Red" "Green" "Green" "Black" "White"  
 [10] "Red" "Red" "White" "Green" "Green" "Black" "Red" "Green" "Green"  
 [19] "White" "Green" "Green" "Green" "Red" "Green" "Red" "Red" "Red"   
 [28] "Red" "White" "Green" "Red" "White" "Black" "Red" "Black" "Black"  
 [37] "Green" "Red" "Black" "Red" "Black" "Black" "Red" "Red" "White"  
 [46] "Black" "Green" "Red" "Red" "Black" "Black" "Red" "White" "Red"   
 [55] "Green" "Green" "Black" "Green" "White" "Black" "Red" "Green" "Green"  
 [64] "White" "Black" "Red" "Red" "Black" "Green" "White" "Green" "Red"   
 [73] "White" "White" "Green" "Green" "Green" "Green" "Green" "White" "Black"  
 [82] "Green" "White" "Black" "Black" "Red" "Red" "White" "White" "White"  
 [91] "White" "Red" "Red" "Red" "White" "Black" "White" "Black" "Black"  
[100] "White"

{: .output}

If we wish to import our string data as chr to allow easy modification and **subsequently** convert a column to a factor to gain error protection, we can use the as.factor() function.

# Import with the default action of treating string data as chr  
carSpeeds <- read.csv(file = 'data/car-speeds.csv')  
str(carSpeeds)

{: .language-r}

'data.frame': 100 obs. of 3 variables:  
 $ Color: chr "Blue" " Red" "Blue" "White" ...  
 $ Speed: int 32 45 35 34 25 41 34 29 31 26 ...  
 $ State: chr "NewMexico" "Arizona" "Colorado" "Arizona" ...

{: .output}

# Change a chr column to a factor  
carSpeeds$Color <- as.factor(carSpeeds$Color)  
str(carSpeeds)

{: .language-r}

'data.frame': 100 obs. of 3 variables:  
 $ Color: Factor w/ 5 levels " Red","Black",..: 3 1 3 5 4 3 3 2 5 4 ...  
 $ Speed: int 32 45 35 34 25 41 34 29 31 26 ...  
 $ State: chr "NewMexico" "Arizona" "Colorado" "Arizona" ...

{: .output}

### The as.is Argument

This is an extension of the stringsAsFactors argument, but gives you control over individual columns. For example, if we want the colors of cars imported as strings, but we want the names of the states imported as factors, we would load the data set as:

carSpeeds <- read.csv(file = 'data/car-speeds.csv', as.is = 1)  
  
# Note, the 1 applies as.is to the first column only

{: .language-r}

Now we can see that if we try to replace ‘Blue’ with ‘Green’ in the $Color column everything looks fine, while trying to replace ‘Arizona’ with ‘Ohio’ in the $State column returns the factor numbers for the names of states that we haven’t replaced:

str(carSpeeds)

{: .language-r}

'data.frame': 100 obs. of 3 variables:  
 $ Color: chr "Blue" " Red" "Blue" "White" ...  
 $ Speed: int 32 45 35 34 25 41 34 29 31 26 ...  
 $ State: Factor w/ 4 levels "Arizona","Colorado",..: 3 1 2 1 1 1 3 2 1 2 ...

{: .output}

carSpeeds$Color <- ifelse(carSpeeds$Color == 'Blue', 'Green', carSpeeds$Color)  
carSpeeds$Color

{: .language-r}

[1] "Green" " Red" "Green" "White" "Red" "Green" "Green" "Black" "White"  
 [10] "Red" "Red" "White" "Green" "Green" "Black" "Red" "Green" "Green"  
 [19] "White" "Green" "Green" "Green" "Red" "Green" "Red" "Red" "Red"   
 [28] "Red" "White" "Green" "Red" "White" "Black" "Red" "Black" "Black"  
 [37] "Green" "Red" "Black" "Red" "Black" "Black" "Red" "Red" "White"  
 [46] "Black" "Green" "Red" "Red" "Black" "Black" "Red" "White" "Red"   
 [55] "Green" "Green" "Black" "Green" "White" "Black" "Red" "Green" "Green"  
 [64] "White" "Black" "Red" "Red" "Black" "Green" "White" "Green" "Red"   
 [73] "White" "White" "Green" "Green" "Green" "Green" "Green" "White" "Black"  
 [82] "Green" "White" "Black" "Black" "Red" "Red" "White" "White" "White"  
 [91] "White" "Red" "Red" "Red" "White" "Black" "White" "Black" "Black"  
[100] "White"

{: .output}

carSpeeds$State <- ifelse(carSpeeds$State == 'Arizona', 'Ohio', carSpeeds$State)  
carSpeeds$State

{: .language-r}

[1] "3" "Ohio" "2" "Ohio" "Ohio" "Ohio" "3" "2" "Ohio" "2"   
 [11] "4" "4" "4" "4" "4" "3" "Ohio" "3" "Ohio" "4"   
 [21] "4" "4" "3" "2" "2" "3" "2" "4" "2" "4"   
 [31] "3" "2" "2" "4" "2" "2" "3" "Ohio" "4" "2"   
 [41] "2" "3" "Ohio" "4" "Ohio" "2" "3" "3" "3" "2"   
 [51] "Ohio" "4" "4" "Ohio" "3" "2" "4" "2" "4" "4"   
 [61] "4" "2" "3" "2" "3" "2" "3" "Ohio" "3" "4"   
 [71] "4" "2" "Ohio" "4" "2" "2" "2" "Ohio" "3" "Ohio"  
 [81] "4" "2" "2" "Ohio" "Ohio" "Ohio" "4" "Ohio" "4" "4"   
 [91] "4" "Ohio" "Ohio" "3" "2" "2" "4" "3" "Ohio" "4"

{: .output} We can see that $Color column is a character while $State is a factor.

## Updating Values in a Factor

Suppose we want to keep the colors of cars as factors for some other operations we want to perform. Write code for replacing ‘Blue’ with ‘Green’ in the $Color column of the cars dataset without importing the data with stringsAsFactors=FALSE.

## Solution

carSpeeds <- read.csv(file = 'data/car-speeds.csv')  
# Replace 'Blue' with 'Green' in cars$Color without using the stringsAsFactors  
# or as.is arguments  
carSpeeds$Color <- ifelse(as.character(carSpeeds$Color) == 'Blue',  
 'Green',  
 as.character(carSpeeds$Color))  
# Convert colors back to factors  
carSpeeds$Color <- as.factor(carSpeeds$Color)

{: .language-r} {: .solution} {: .challenge}

### The strip.white Argument

It is not uncommon for mistakes to have been made when the data were recorded, for example a space (whitespace) may have been inserted before a data value. By default this whitespace will be kept in the R environment, such that ‘ Red’ will be recognized as a different value than ‘Red’. In order to avoid this type of error, use the strip.white argument. Let’s see how this works by checking for the unique values in the $Color column of our dataset:

Here, the data recorder added a space before the color of the car in one of the cells:

# We use the built-in unique() function to extract the unique colors in our dataset  
unique(carSpeeds$Color)

{: .language-r}

[1] Green Red White Red Black  
Levels: Red Black Green Red White

{: .output}

Oops, we see two values for red cars.

Let’s try again, this time importing the data using the strip.white argument. NOTE - this argument must be accompanied by the sep argument, by which we indicate the type of delimiter in the file (the comma for most .csv files)

carSpeeds <- read.csv(  
 file = 'data/car-speeds.csv',  
 stringsAsFactors = FALSE,   
 strip.white = TRUE,  
 sep = ','  
 )  
  
unique(carSpeeds$Color)

{: .language-r}

[1] "Blue" "Red" "White" "Black"

{: .output}

That’s better!

### Write a New .csv and Explore the Arguments

After altering our cars dataset by replacing ‘Blue’ with ‘Green’ in the $Color column, we now want to save the output. There are several arguments for the write.csv(...) [function call](%7B%7B%20page.root%20%7D%7D/reference.html#function-call), a few of which are particularly important for how the data are exported. Let’s explore these now.

# Export the data. The write.csv() function requires a minimum of two  
# arguments, the data to be saved and the name of the output file.  
  
write.csv(carSpeeds, file = 'data/car-speeds-cleaned.csv')

{: .language-r}

If you open the file, you’ll see that it has header names, because the data had headers within R, but that there are numbers in the first column.

### The row.names Argument

This argument allows us to set the names of the rows in the output data file. R’s default for this argument is TRUE, and since it does not know what else to name the rows for the cars data set, it resorts to using row numbers. To correct this, we can set row.names to FALSE:

write.csv(carSpeeds, file = 'data/car-speeds-cleaned.csv', row.names = FALSE)

{: .language-r}

Now we see:

## Setting Column Names

There is also a col.names argument, which can be used to set the column names for a data set without headers. If the data set already has headers (e.g., we used the headers = TRUE argument when importing the data) then a col.names argument will be ignored. {: .callout}

### The na Argument

There are times when we want to specify certain values for NAs in the data set (e.g., we are going to pass the data to a program that only accepts -9999 as a nodata value). In this case, we want to set the NA value of our output file to the desired value, using the na argument. Let’s see how this works:

# First, replace the speed in the 3rd row with NA, by using an index (square  
# brackets to indicate the position of the value we want to replace)  
carSpeeds$Speed[3] <- NA  
head(carSpeeds)

{: .language-r}

Color Speed State  
1 Blue 32 NewMexico  
2 Red 45 Arizona  
3 Blue NA Colorado  
4 White 34 Arizona  
5 Red 25 Arizona  
6 Blue 41 Arizona

{: .output}

write.csv(carSpeeds, file = 'data/car-speeds-cleaned.csv', row.names = FALSE)

{: .language-r}

Now we’ll set NA to -9999 when we write the new .csv file:

# Note - the na argument requires a string input  
write.csv(carSpeeds,  
 file = 'data/car-speeds-cleaned.csv',  
 row.names = FALSE,  
 na = '-9999')

{: .language-r}

And we see:

{% include links.md %}