Week 9 Module 2 – Reading and Writing Data Frames

CSCI E-5a: Programming to R

Let's clear the global computing environment:

```
rm( list = ls() )
```

Module Overview and Learning Outcomes

Hello! And welcome to Module 2: Reading and Writing Data Frames.

In this module, we're going to learn how to read and write data files.

- In Section 1, we'll learn a simple method for reading data in from a file.
- In Section 2, we'll learn a simple method for writing data out to a file.
- In Section 3, we'll consider the question of what R does with character vectors when constructing a data frame.
- In Section 4, we'll explore how to use "paths" to specify files that are not in the same folder as the currently running R notebook.

When you've finished this module, you should be able to:

- Read in data in either .csv or tab-delimited formats.
- Write out data in either .csv or tab-delimited formats.
- Explain how R traditionally handled character string vectors when constructing a data frame, and how this default behavior has recently changed.
- Use the concept of a path to access files that are not in the same folder as the current R notebook.

Also, we'll meet three new built-in R functions in this module:

- `read.csv()`
- `write.csv()`
- `getwd()`

Allright, let's get rolling!

Section 1: Reading Data In From A File

Main Idea: *We can read in data from a file*

In this section, we'll learn a simple method for reading data in from a file.

There are many file formats i.e. different ways of storing data in a file, and each will have its own particular set of features.

Indeed, the whole field of database management is concerned with this topic: how can we efficiently store data in a file?

This is a huge topic, and I'm not going to try cover all the possible ways to do this.

There are so many different data file structures that we can't study each one, so I'm going to show you a very simple system that will get you started, and will teach you the basic ideas of working with files.

If you're in a situation where you need to read in a specialized type of file format, you should be able to read through the documentation and make any necessary changes.

And even though the system that I will show you is simple and easy to understand, it is nonetheless very popular in the real world.

In this approach to storing data in a file, we start with a rectangular data array:

Age	Likes Ice Cream
61	TRUE
43	FALSE
27	TRUE
38	TRUE
52	FALSE

Here we have 2 variables, `Age` and `Likes Ice Cream`, and 5 experimental units, which in this case are individual people.

To store this data in a "comma separated values" file format, each row is stored on a single line of a text file, and each variable is separated from any others by a comma.

Thus, the text of a comma-separated-values text file that stores our ice cream dataset is:

61,TRUE 43,FALSE 27,TRUE 38,TRUE 52,FALSE

Although this is an acceptable way to store the data, it's usually better to include the names of the variables in the first line:

Age,Likes Ice Cream 61,TRUE 43,FALSE 27,TRUE 38,TRUE 52,FALSE

Notice that the names of the variables are also separated by commas, just like the actual data values.

We can then read this data into R with the command:

```
read.csv( "LikeIceCream.csv" )
```

```
##   Age Likes.Ice.Cream
## 1  61            TRUE
## 2  43           FALSE
## 3  27            TRUE
## 4  38            TRUE
## 5  52           FALSE
```

Notice that in the original text file the variable name was written with spaces as "Likes Ice Cream", but R does not allow spaces in a variable name, so it substitutes periods to obtain "Likes.Ice.Cream".

Of course, it's probably a good idea to store this data frame into a variable so we can work with it later:

```
ice.cream.dataset <-
    read.csv( "LikeIceCream.csv" )

head( ice.cream.dataset )
```

```
##   Age Likes.Ice.Cream
## 1  61            TRUE
## 2  43           FALSE
## 3  27            TRUE
## 4  38            TRUE
## 5  52           FALSE
```

Another way to store data in a text file is to use a tab stop as the delimiter instead of a comma.

These are called "tab-delimited" files, and they are easy to create in Microsoft Excel.

To read in a tab-delimited file, we can still use the `read.csv()` function, but now we have to specify the delimiter is a tab stop, which we can write as "\t":

```
ice.cream.data.frame.2 <-
    read.csv( "LikeIceCream.txt", sep = "\t" )

ice.cream.data.frame.2
```

```
##   Age Likes.Ice.Cream
## 1  61            TRUE
## 2  43           FALSE
## 3  27            TRUE
## 4  38            TRUE
## 5  52           FALSE
```

So that's how to read in data from a file.

Now let's see how to write data out to a file.

Section 2: Writing Data to a File

> **Main Idea:** *We can write data out to a file*

In this section, we'll learn a simple method for writing data out to a file.

Of course, just as it's useful to read data in from a file, it's also useful to be able to write a data frame out to a file.

We can use the `write.csv()` function to write out a CSV file:

```
write.csv(
    x = ice.cream.dataset,
    file = "Ice Cream Dataset.csv"
)
```

The `write.csv()` function does NOT have an option to allow you to select the separator, so if you want to write out a tab-delimited file you have to use `write.table()`:

```
write.table(
    x = ice.cream.dataset,
    file = "Ice Cream Dataset Tab.txt",
    sep = "\t"
)
```

Warning: if you write a data frame out to a CSV file, then all factors are converted to their string labels and you lose all the internal information in a factor such as pre-specified levels and labels.

So that's how to write data out to file.

Now let's explore how R handles factors when reading in data frames.

Section 3: Character Strings and Data Frames

> **Main Idea:** *R can convert character string values to a factor when reading in data*

In this section, we'll consider the question of what R does with character string values when constructing a data frame.

So far we've generated factors by hand, starting with a vector and then calling the `factor()` function.

For instance, let's create a data frame with a character vector in it:

```
enzyme.data.frame <-
    data.frame(
        enzyme.level =
            c(15, 12, 22, 25, 17),
        species =
            c( "Hedgehog", "Hedgehog",
               "Armadillo", "Hedgehog",
               "Armadillo"
            )
    )

enzyme.data.frame
```

```
##   enzyme.level   species
## 1           15  Hedgehog
## 2           12  Hedgehog
## 3           22 Armadillo
## 4           25  Hedgehog
## 5           17 Armadillo
```

What kind of object is the `species` column vector in this data frame?

It's just a character vector:

```
class( enzyme.data.frame$species )
```

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

However, by using the `stringsAsFactors` option, we can have R automatically convert this character string into a factor when constructing the data frame:

```
factor.data.frame <-
    data.frame(
        enzyme.level =
            c(15, 12, 22, 25, 17),
        species =
            c( "Hedgehog", "Hedgehog",
               "Armadillo", "Hedgehog",
               "Armadillo"
            ),
        stringsAsFactors = TRUE
    )

factor.data.frame
```

```
##   enzyme.level   species
## 1           15  Hedgehog
## 2           12  Hedgehog
## 3           22 Armadillo
## 4           25  Hedgehog
## 5           17 Armadillo
```

Now let's take a look at the class of the `species` column vector:

```
class( factor.data.frame$species )
```

```
## [1] "factor"
```

In fact, for a long time, this was the standard way of encountering factors, because R traditionally used a default value of `TRUE` for the `stringsAsFactors` option.

Thus, whenever R read in a data frame with character vectors, it would always automatically convert them into factors.

However, with the release of R 4.0, this has changed, and now the default value for `stringsAsFactors` is `FALSE`.

So now, with R 4.0, character strings are *not* automatically converted into factors, and if you want R to do this you have to explicitly override the default setting.

This strikes me as a bad idea, because it means that there will be many scripts than ran perfectly fine under earlier versions of R, but won't now that character strings are not converted to factors.

Thus, code that used to be correct is now broken.

This is something to bear in mind if you're working with old code – the default behavior is now different, and that old code might not work properly.

Personally, I like factors, and in my experience most character vectors in a data frame are usually functioning as factors, so I like the old default setting where these were automatically converted to factors.

On the other hand, it's not hard to convert a character string vector into a factor, so this isn't really a big deal.

However, it *is* a big deal that old code now no longer works properly, and this is a subtle bug, so just be aware of this issue.

So that's how R handles character string values when reading in a data frame.

Now let's learn how to specify a path in the file system.

Section 4: Paths

> **Main Idea:** *We can specify a path through the file system*

In this section, we'll explore how to use "paths" to specify files that are not in the same folder as the currently running R notebook.

So far, whenever we've worked with files, I've said that they have to be in the same folder as the R notebook that you're using.

This isn't entirely correct.

It's possible to access files in other locations than the current R notebook folder, but in order to do this you have to understand the concept of "paths".

It's always a little difficult for me to approach the topic of "paths", because some people understand them very well, and some people have never encountered the concept before.

First of all, when you start an R notebook, it automatically sets your current location to the folder where the notebook is stored.

We can find out the current location using the `getwd()` function, which takes no arguments:

```
getwd()
```

```
## [1] "C:/Users/Harvard/Documents/Abc/CSCI E-5a Spring 2022/Week 9 -- Data Frames/Week 9 ZIP Bundle/Mo
```

There are two simple ideas behind constructing a path:

- To go up to the parent folder, use double dots "..", followed by a forward slash.
- To go into a subfolder, use a forward slash, followed by the name of the subfolder.

Let's see an example.

Currently, we are in the folder "Module 2 – Reading and Writing Data Files", which is itself contained in the folder "Modules".

We would like to open the file "Problem 1 Data.csv", which is located in a folder called "Problem Set 8 Data", which is itself contained in a folder called "Problem Set 8" of the Unit 9 folder.

So, to construct the path for the file "Problem 1 Data.csv", starting from the current location of "R Notebook" in "Module 2 – Reading and Writing Data Files", we take this path:

- First, we go up a level to the "Modules" folder, and so we use the "../" construct.
- Then we go up another level to the "Unit 9 – Data Frames" folder, so that's one more use of the "../" construct.
- Then we go down into the "Problem Set 8" folder.

- Then we go down into the "Problem Set 8 Data" folder.
- Finally we can access the file using its name.

Here's the full path:

```
problem.1.data <-
  read.csv(
    file = "../../Problem Set 8/Problem Set 8 Data/Problem 1 Data.csv"
  )

head( problem.1.data )
```

```
##   enzyme.1 high.credit.status   species
## 1       94               TRUE  Hedgehog
## 2      102               TRUE  Aardvark
## 3       92               TRUE Armadillo
## 4      116               TRUE Armadillo
## 5      103               TRUE Armadillo
## 6       92               TRUE  Platypus
```

So that' how to specify a path in the file system.

Now let's review what we've learned in this module.

Module Review

In this module, we learned how to read and write data files.

- In Section 1, we learned a simple method for reading data in from a file.
- In Section 2, we learned a simple method for writing data out to a file.
- In Section 3, we considered the question of what R does with character vectors when constructing a data frame.
- In Section 4, we explored how to use "paths" to specify files that are not in the same folder as the currently running R notebook.

Now that you've finished this module, you should be able to:

- Read in data in either .csv or tab-delimited formats.
- Write out data in either .csv or tab-delimited formats.
- Explain how R traditionally handled character string vectors when constructing a data frame, and how this default behavior has recently changed.
- Use the concept of a path to access files that are not in the same folder as the current R notebook.

Also, we met three new built-in R functions in this module:

- `read.csv()`

- `write.csv()`
- `getwd()`

Allright, that's it for Module 2: Reading and Writing Data Frames.

Now let's move on to Module 3: Selecting Columns.

See you there!