5 - File Input and Output

R is a platform for analyzing real data, so one of the basic things that it has to do well is import data from files and databases into its data structures, as well as export its data to files. R does this through file input/output functions.

RStudio can import data files into data structures through the Files menu -> Import Dataset function, which is good for previewing how the data would be imported. To load large data sets, R's ***read*** family of functions should be used. See ?read.table or ?read.csv.

read.table() function

The read.table() function is the principal means of reading tabular data into R, and the other read.*() functions are based on it. Depending on the format and file system location of the input data file, using the read.table() can be as simple as

```
mydataframe <- read.table('data_table.txt')
```

The first argument to the read.table() function is the quoted file name. If the data file is in the current working directory (folder) where R was started, then just the file name is enough. But if the data file is in some other folder, read.table() should be given the absolute file system pathname of the data file, e.g.,

```
mydataframe <- read.table('C:/Users/BFHit/CS201/data/data_table.txt') # on Windows

mydataframe <- read.table('/Users/BFHit/CS201/data/data_table.txt') # on macOS
```

Folder management

The getwd() function in R will give you the value of R's current working directory (folder).

```
# working directory on CSSE dept Linux account
getwd()
```

```
[1] "/export/home/hawkdom2/jchung/classes/cs201/sp23/data"
```

The setwd() function in R will set R's current working directory.

```
# on university CSSE dept Linux account
getwd()
[1] "/export/home/hawkdom2/jchung"

setwd('/export/home/hawkdom2/jchung/cs201/data')
getwd()
[1] "/export/home/hawkdom2/jchung/public_html/cs201/sp23/data"
```

Some other folder management functions are as follows.

```
dir()               - show files in current folder
file.create()
file.rename()
file.remove()       - deletes a file
file.copy()
file.show()         - show file contents using a pager (hit q to quit)
dir.create()        - creates folder
file.exists()       - returns logical (TRUE or FALSE)
dir.exists()
```

Other read.table() options (input params)

```
# not the default values for these options;
# see ?read.table for default values
#
header = TRUE             - read first line as column names
dec = ","                 - change to comma as decimal indicator
sep = "_"                 - change column separator to underscore
fill = TRUE               - fill incomplete rows with NAs at the end
skip = 12                 - ignore the first 12 lines (e.g. with meta data)
comment.char = "%"        - omit lines that start with % (like R's #)
na.strings = c(-999, "NN")      - identify "-999" and "NN" as NA entries (missing values)
stringsAsFactors = TRUE   - do or do not convert characters to factors
```

Expected data file format

To read an entire data frame directly, the external file will normally have a special format. The first line of the file should have a name for each variable in the data frame. Each additional line of the file has as its first item a row label and the values for each variable. An example of this format is gmp.dat (gross metropolitan product data).

```
"MSA" "gmp" "pcgmp"
"1" "Abilene, TX" 3.887e+09 24490
```

```
"2" "Akron, OH" 2.2998e+10 32889
"3" "Albany, GA" 3.955e+09 24269
"4" "Albany-Schenectady-Troy, NY" 3.1321e+10 36836
"5" "Albuquerque, NM" 3.0727e+10 37657
"6" "Alexandria, LA" 3.879e+09 25494
"7" "Allentown-Bethlehem-Easton, PA-NJ" 2.3964e+10 30165
"8" "Altoona, PA" 3.39e+09 27008
"9" "Amarillo, TX" 7.041e+09 29360
...
```

The row labels here are "1", "2", "3", etc. The double quotes around certain values is not needed. It makes sense to have double quotes around the MSA values because they consist of multiple words that need to be consider one value, e.g., MSA = "Abilene, TX". Numeric values will not be quoted.

Exporting data

The write.table() function can be used to take an R data frame and create a data file from it. The data file that is created shows the expected data file format. See ?write.table.

Use the write.table() function to create a data file from the R built-in mtcars data frame.

```
# this creates mtcars.dat in the current working directory (folder)
write.table(mtcars, 'mtcars.dat')

# this creates mtcars.dat in some other working directory
write.table(mtcars, 'C:/Users/JoeC/Desktop/mtcars.dat')
```

If you open the mtcars.dat file in notepad or other text editor, it should look like the following, where the row labels are the car names (make and model), and the values are separated by 1 space.

```
"mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear" "carb"
"Mazda RX4" 21 6 160 110 3.9 2.62 16.46 0 1 4 4
"Mazda RX4 Wag" 21 6 160 110 3.9 2.875 17.02 0 1 4 4
"Datsun 710" 22.8 4 108 93 3.85 2.32 18.61 1 1 4 1
"Hornet 4 Drive" 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1
"Hornet Sportabout" 18.7 8 360 175 3.15 3.44 17.02 0 0 3 2
...
```

The write.csv() function can be used to create a comma-separated value (CSV) file from a data frame. The output format is almost the same, with the values separated by a comma.

```
# this creates mtcars.csv in the current working directory (folder)
write.csv(mtcars, 'mtcars.csv')

# what the comma-separated values look like
file.show('mtcars.csv')
"","mpg","cyl","disp","hp","drat","wt","qsec","vs","am","gear","carb"
```

```
"Mazda RX4",21,6,160,110,3.9,2.62,16.46,0,1,4,4
"Mazda RX4 Wag",21,6,160,110,3.9,2.875,17.02,0,1,4,4
"Datsun 710",22.8,4,108,93,3.85,2.32,18.61,1,1,4,1
"Hornet 4 Drive",21.4,6,258,110,3.08,3.215,19.44,1,0,3,1
"Hornet Sportabout",18.7,8,360,175,3.15,3.44,17.02,0,0,3,2
...
```

Dealing with missing values

Missing values are common in real data. If a data file is missing certain values in a row of values, the read.table() function will normally print an error message, and the data import will fail. For example, a mtcars_missing_vals.dat has some missing values for two cars:

```
mtcars_missing_vals <- read.table('mtcars_missing_vals.dat')
Error in scan(file = file, what = what, sep = sep, quote = quote, dec = dec,  :
  line 2 did not have 12 elements
```

To read the data anyway with the missing values, the 'fill' option to read.table() can be used to add the NA value to the end of missing value rows.

```
mtcars_missing_vals <- read.table('mtcars_missing_vals.dat', fill=TRUE)
```

The NA values at the end of rows indicate rows that should be fixed before using the result data frame.

If the missing values in the data file are replaced with "NA", then read.table() will be able to deal with the missing values more gracefully. But not all data files are so nice.

CSV files that are read with the read.csv() function deal with missing values more gracefully. If the mtcars.csv has missing values in certain rows, commas will be bunched together to indicate missing values.

```
"","mpg","cyl","disp","hp","drat","wt","qsec","vs","am","gear","carb"
"Mazda RX4",21,6,160,110,3.9,2.62,16.46,0,1,4,4
"Mazda RX4 Wag",21,6,160,110,3.9,2.875,,0,1,4,4
"Datsun 710",22.8,4,108,93,3.85,2.32,18.61,1,1,4,1
"Hornet 4 Drive",21.4,6,258,110,3.08,3.215,,1,0,3,1
...
```

When the CSV file is read with read.csv(), the NAs show up in the correct columns.

```
mtcars_missing_vals <- read.csv('mtcars_missing_vals.csv')
                    X  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
1           Mazda RX4 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
2       Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875    NA  0  1    4    4
3          Datsun 710 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
4      Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215    NA  1  0    3    1
...
```

File Examples

Example 1: animals.txt (on Piazza -> General Resources)

```
# Read the file animals.txt with the correct settings for each needed argument
# and assign it to an object (data frame called animals).
animals <- read.table('animals.txt', sep = '\t', header=TRUE, dec=',')

# Check whether everything is read correctly with str.
str(animals)

# Print only the first column. Try this in 3 different ways of subsetting.
animals$Name
animals[,1]         # using matrix notation
animals[,'Name']    # ditto

# which element of this vector (animals$Name) is equal to (==) "Sparrow"?
which(animals$Name == 'Sparrow')

# Can the elephant fly? Find out with a logical query (3 different ways).
animals[4,3]
animals[animals$Name == 'Elephant', 3]
animals[animals$Name == 'Elephant', 'can.fly']

# Find out how many legs the mammals have on average. Interpret the result.
animals[animals$Class == 'Mammal', 'n.legs']
mean(animals[animals$Class == 'Mammal', 'n.legs'])

# (Advanced)
# Use the tapply() function to find the average cuteness score for each animal species (Class).
tapply(animals$cuteness, INDEX=animals$Class, FUN=mean)
```

Example 2: english.sorted (on Piazza -> General Resources)

In this example, we want to read a list of 121,661 words in english.sorted into a data frame and then find the number of occurrences of each word length in the word list.

```
# Read the file english.sorted with the correct settings for each needed argument
# and assign it to a data frame. The english.sorted word list has no column header,
# so using the header=FALSE option below would be needed.
english.sorted <- read.table('english.sorted', header=FALSE)

# See structure of english.sorted.
head(english.sorted)
str(english.sorted)

# Get the word lengths for all words using a for loop and nchar() function.
word_lengths <- vector('integer', length(english.sorted$V1)) # create empty 121,661-element vector
```

```
for (i in 1:length(english.sorted$V1)) {
      # in the word_lengths vector at index i, store the number of characters in word i
      word_lengths[i] <- nchar(english.sorted$V1[i])
}

# Alternatively, get the word lengths vector using the sapply() and nchar() functions.
word_lengths <- sapply(english.sorted$V1, nchar)

# Create contingency table to get word length frequencies in word_lengths.
word_length_frequencies <- table(word_lengths)

# Generate bar plot of word length frequencies.
barplot(word_length_frequencies, xlab='Word lengths', ylab='Occurrences')

# Show word_length_frequencies in tabular format by making a data frame from it and renaming
# the columns using the names() function.
word_length_freq_dframe <- as.data.frame(word_length_frequencies)
names(word_length_freq_dframe) <- c('Word Lengths', 'Occurrences')
print(word_length_freq_dframe, row.names = FALSE)


# (Advanced)
# Notice there are no occurrences of 26-character words in the word_length_frequencies table.
# This also shows up in the barplot.
word_length_frequencies

# Contingency tables are made up of factors and associated values. We need to make
# the "26" factor show up in the word_length_frequencies table even though there are no #
26-character words in english.sorted.
?factor

# We'll need to use the 'levels' option in the factor() function to make sure that all word
# lengths from 1 to max_length=28 are represented in the contingency table.
max_length <- max(word_lengths)
factor(word_lengths, level=1:max_length)
table(factor(word_lengths, level=1:max_length))
word_length_frequencies <- table(factor(word_lengths, level=1:max_length))
```

Example 3: 2020-2023 US State population estimates (on Piazza -> General Resources)

In this example, we want to download an Excel spreadsheet of 2020-2023 US State population estimates and convert it to a CSV file called 2020_2023_state_region_pop.csv. You'll need to use a spreadsheet to export the spreadsheet to CSV. After you have saved 2020_2023_state_region_pop.csv, do the following:

Read 2020_2022_state_region_pop.csv into a data frame called state.pop. Examine state.pop with str(). Use the state.pop data frame for the remainder of this assignment.

```
      state.pop <- read.table("2020_2022_state_region_pop.csv", sep=",", header=TRUE, dec=".")
```

```
                    # header=TRUE and dec="." are the default parameters, so are optional.

      # or more simply,

      state.pop <- read.csv("2020_2023_state_region_pop.csv")
```

Optional: Use attach() on state.pop so that you can access the columns of state.pop without having to type state.pop$ in front of the column names.

Write code to list the names of the states that decreased in population from 2020-2023, i.e., POPESTIMATE2023 - POPESTIMATE2020 < 0.

```
      state.pop$NAME[which(state.pop$POPESTIMATE2023 - state.pop$POPESTIMATE2020 < 0)]

      # or just

      state.pop$NAME[state.pop$POPESTIMATE2023 - state.pop$POPESTIMATE2020 < 0]
```

Write code to output the names of the five states that had the greatest increase in population from 2020-2023. Include the population increase numbers in the output.

```
      # Create vector of population changes 2020-2023 and add it as a new data frame column:
      pop.diff <- state.pop$POPESTIMATE2023 - state.pop$POPESTIMATE2020
      state.pop$POP_DIFF <- pop.diff # becomes column 7 of state.pop

      # Order data frame by sorted POP_DIFF, descending order;
      # show only the state NAME and POP_DIFF columns (columns 2 and 7)
      state.pop[order(state.pop$POP_DIFF, decreasing=TRUE), c(2, 7)]

      # Restrict to top five states (first 5 rows):
      state.pop[order(state.pop$POP_DIFF, decreasing=TRUE), c(2, 7)][1:5,]
      # or
      head(state.pop[order(state.pop$POP_DIFF, decreasing=TRUE), c(2, 7)], n=5)
```

Write code to output the names of the five states that had the greatest decrease in population from 2020-2023. Include the population decrease numbers in the output.

```
      # Same as above but use decreasing=FALSE.
```

Write code to find the change in population for each of the four ***regions*** from 2020-2023.

```
      # Get sum of POP_DIFF for the Northeast Region states:
      sum(state.pop[state.pop$REGION == 'Northeast', ]$POP_DIFF)

      # Repeat for the other 3 regions

      # Or use tapply:
      tapply(state.pop$POP_DIFF, INDEX=state.pop$REGION, FUN=sum)
```