

File Operations

Table of Content

Introduction	3
1. File Operations.....	4
1.1 Reading Data using the scan() Function	4
1.2 Reading a CSV file.....	7
1.3 Reading from Text Files	9
1.4 Writing Data to CSV Files (comma separated value file)	10
1.5 Writing to Text Files	12
1.6 Loading Data from an XML File into an R Object	13
Summary	16

Introduction

This topic deals with file I/O operations using R. Large data sets are typically saved as text, CSV, and XML files. Analysis of data stored in the files involves importing or reading the data from the files, storing them into R objects, and manipulating them.

The contents of an R object would be saved into files on several occasion. This is handled using the file write operations in R. In general, the file I/O operations forms an integral part of data analytics using R.

Learning Objectives

Upon completion of this topic, you will be able to:

- Explain the method of exporting contents of a data frame into a file
- Discuss the process of importing contents of a CSV file into a data frame
- Explain the method of importing contents of a text file into a data frame
- Discuss the steps to import contents of an XML document into a data frame

1. File Operations

Data or information is collected and stored into various files to use whenever required. The different file I/O operations that can be performed on these stored files are:

Reading data from
the console using the
`scan()` function

Reading from CSV
files

Reading from text
files

Writing to CSV files

Writing to text files

Importing XML files

1.1 Reading Data using the `scan()` Function

The `scan()` command is used to read data from the console directly.

Pseudocode:

```
scan()
```

The entered data is stored in a new variable in the following way:

```
newdata <- scan() # On pressing the enter key, R console prompts to #  
#the user enter the data
```

Example:

```
z<-scan()  
1: 10 11  
3
```

Output:

Read 2 items, here the first item saved into "z" is a vector and the second item saved is 11 which is read from the console s

Entering text as data through the scan() Command:

- **scan()** reads numeric data by default. However, it can also read character data using the **what** option.
- **what** option should be specified as **what=character()** or **what=" "** and all the fields will be read as strings.

Example:

```
myscan = scan(what='character')  
1: hi hello  
3: good morning  
5:
```

Output:

```
Read 4 items  
> myscan # printing the contents of the variable "myscan"  
[1] "hi"      "hello"   "good"    "morning"
```

The example above has read a set of inputs as strings.

Copying Data using Clipboard (when directly copying from excel to R):

When directly copying values from CSV file (which uses a comma to separate values) to R the command **sep=','** is used.

Example:

```
data=scan(sep =', ')  
1: 10,11,12,3,1,2,4,5,6,7,8,89  
13:
```

Output:

```
Read 12 items  
> data  
[1] 10 11 12 3 1 2 4 5 6 7 8 89
```

In the above example, a set of numbers are passed through the console, which is separated by a comma and is assigned to a variable “data”. The variable then prints the values in R.

Reading a file from a disk using scan

The **scan()** command can be used to read data from a file. To read a file from the console, the below syntax is used:

Syntax:


```
myscan=scan(file='filename.txt')
```

Example:

```
myscan = scan(file='sampletext.txt',what='character')
```

Output:

```
Read 1 item  
> myscan  
[1] "hello"
```




Note: Inside the scan command, the file path must be enclosed in quotes.

1.2 Reading a CSV file

The `read.csv` command is used for reading a large amount of data stored in a CSV file. Below is a list of additional arguments that can be added to the basic syntax of reading a CSV file.

```
read.csv() # The read.csv will search for a csv file and reads it  
into R.  
  
read.csv(file,sep=',',header=TRUE,row.names)  
  
read.csv(file,sep=',',header = TRUE,row.names=FALSE)  
  
read.csv("<file.choose()>")
```



Note: The **file.choose** option would allow the user to select a file from a pop-up window, allowing the user to select a file to be read from the file system.

file	The file path
header	By default, the first row of the file is considered as a header unless this option is set to false (header=FALSE)
row.names	This argument specifies the row names for the data
sep	This is used to mention the field delimiter in the file

Example:

```
samp.data=read.csv(file.choose())  
samp.data)
```

Output:

```
Pre.training Post.Training  
1           18           22  
2           21           25  
3           16           17  
4           22           24  
5           19           16  
6           24           29  
7           17           20  
8           21           23  
9           23           19  
10          18           20  
11          14           15  
12          16           15  
13          16           18
```


1.3 Reading from Text Files

Read.table() is a basic R command for reading data from text files, which is a more generic way to read/write tabular data from/to disk.

Pseudocode:

```
data <- read.table('<filename>'
                  , header = F
                  , fill = T
                  , sep="\t" )
```

filename	The file path
header	The default option for read.table is to consider the first line of the file as a header. It can be changed by specifying header=FALSE
Fill = T	This option allows the blank fields to be added when rows of unequal lengths are present in the file that is being imported
sep="\\t"	This indicates that it is a tab delimited file

Example:

```
read.table(file.choose(), header=TRUE, sep=', ')
```

	Subject.A	Subject.B	Subject.C
1	10	50	40
2	20	40	50
3	30	30	20
4	12	20	13
5	15	10	10
6	16	15	25

The above example demonstrates how a data in a file saved on the disk is read into an R using `read.table()`

1.4 Writing Data to CSV Files (comma separated value file)

Writing a data frame to a CSV file on the disk:

Pseudocode:

```
write.csv(<Data frame name>, "<path to write>/<file_name>.csv")
```

Example: Built-in data frame `mtcars` is written to a CSV file on the disk.

```
write.csv(mtcars, file="newmtcars.csv")
```

A file named `newmtcars.csv` will be created in the current working directory of R and contents of `mtcars` dataset will be written into it, along with the row and the column identifiers.

Omitting the row names in the CSV by setting the `row.names = FALSE`

Example: Writing the contents of `data.df` (a user defined data frame) into a CSV file.

The contents of `data.df` data frame is as follows:


	d1	d2	d3
1	Name1	11.11	47.87
2	Name2	150.10	-40.40
3	Name3	-50.20	-3.50
4	Name4	-100.30	49.80

```
write.csv(data.df, file="newdatadf.csv", row.names = FALSE)  
data.df
```

```
write.csv(data.df, file="newdatadf.csv", row.names=FALSE)
```

Below is the contents of the file **newdatadf.csv** after the write operation in the above statement.

d1	d2	d3
Name1	11.11	47.87
Name2	150.1	-40.4
Name3	-50.2	-3.5
Name4	-100.3	49.8



Note: The row names (1 2 3 4 5) have been omitted

Omitting the NA values while saving the content of a data frame in a CSV file

Example: Consider the contents of a user defined data frame “df” which consists of NA values

```
write.csv(df, file = "dfnew.csv", row.names=FALSE, na="")
```


```
df
```

	a	b	c
1	1	NA	11
2	2	NA	12
3	3	NA	13
4	4	NA	14
5	5	NA	15

The code demonstrates the scenario of having to omit NA

```
write.csv(df, file = "dfnew.csv", row.names=FALSE, na="")
```

In the example above, df is a data frame that will be written to **dfnew.csv** on the disk, omitting the NA values.



Note: Setting the **col.names=FALSE** would omit the column names just like row names are omitted.

1.5 Writing to Text Files

write.table() command is used to write into text files.

Example:


```
write.table(mydata, "c:/mydata.txt", sep="\t")
```

In the above example, the data frame which is to be saved as a text file is “mydata” and the path where it has to be saved is “c:/mydata.txt”. The option sep=“\t” means, it is supposed to be saved as a tab separated file.

1.6 Loading Data from an XML File into an R Object

Example:

```
<?xml version="1.0"?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications
      with XML.</description>
  </book>
  <book id="bk102">
    <author>Ralls, Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
    <description>A former architect battles corporate zombies,
      an evil sorceress, and her own childhood to become queen
      of the world.</description>
  </book>
</catalog>
```



Note: XML files can be downloaded from the internet and saved in an appropriate location.

The required packages should be installed in R, prior to loading XML data. The below are the sequence of commands to install the `xlsxjars`, `xlsx`, and `rJava` packages. This would require an active internet connection.

```
install.packages("rJava") #installs the rJava package
install.packages("xlsx") #installs the xlsx package
install.packages("xlsxjars") #installs the xlsxjars package
```

While trying to install a package in R, messages shown in the below screenshot will be displayed.

```
> install.packages("xlsx")
Installing package into 'C:/Users/vinodr/Documents/R/win-library/3.4'
(as 'lib' is unspecified)
also installing the dependencies 'rJava', 'xlsxjars'

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.4/rJava_0.9-8.zip'
Content type 'application/zip' length 717100 bytes (700 KB)
downloaded 700 KB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.4/xlsxjars_0.6.1.zip'
Content type 'application/zip' length 9485454 bytes (9.0 MB)
downloaded 9.0 MB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.4/xlsx_0.5.7.zip'
Content type 'application/zip' length 401163 bytes (391 KB)
downloaded 391 KB


package 'rJava' successfully unpacked and MD5 sums checked
package 'xlsxjars' successfully unpacked and MD5 sums checked
package 'xlsx' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
  C:\Users\vinodr\AppData\Local\Temp\RtmpmUs6CC\downloaded_packages
> |
```

After installing the package, libraries should be loaded.

```
library(rJava)
library(xlsx)
library(xlsxjars)
library(methods) #This package would already be pre-installed in RStudio
```

```
mydf=xmltoDataFrame("File path") #Mention the path of the XML file where it is saved
```



Note: The function “xmltoDataFrame” loads the contents of the XML file into a data frame in R.

Summary

The different file I/O operations that can be performed on stored files are:

- Reading data from the console using the `scan()` function
- Reading from CSV files
- Reading from text files
- Writing to CSV files
- Writing to text files
- Importing XML files

The **`read.csv`** command is used for reading a large amount of data stored in a CSV file.

`Read.table()` is a basic R command for reading data from text files, which is a more generic way to read/write tabular data from/to disk.

`write.table()` command is used to write into text files.