02 Data Import and Manipulation

June 19, 2020

Contents

1 Introduction

In this notes, examples concerning data processing are presented

2 Read and Write Data Files

2.1 CSV Files

2.1.1 Read CSV Files

The Syntax

```
read.csv(file, header = TRUE, sep = ",", quote = "\"",
              dec = ".", fill = TRUE, comment.char = "", ...)
```

Read a CSV File with Default Options

- `read.csv` returns a data.frame

The comma-separated values (CSV) file used here for demonstration has a content:

```
"","BPchange","Dose","Run","Treatment","Animal"
"1",0.5,6.25,"C1","Control","R1"
"2",4.5,12.5,"C1","Control","R1"
"3",10,25,"C1","Control","R1"
"4",26,50,"C1","Control","R1"
"5",37,100,"C1","Control","R1"
"6",32,200,"C1","Control","R1"
```

[1]:
```
# Getting data from file "rabbit.csv"
rabbit_sample <- read.csv("datasets/rabbit.csv")

# Print the class of the variable rabbit_sample
print(class(rabbit_sample))

# Printing first few lines of the dataframe
head(rabbit_sample)
```

```
[1] "data.frame"
```

A data.frame: 6 × 6

	X <int>	BPchange <dbl>	Dose <dbl>	Run <chr>	Treatment <chr>	Animal <chr>
1	1	0.5	6.25	C1	Control	R1
2	2	4.5	12.50	C1	Control	R1
3	3	10.0	25.00	C1	Control	R1
4	4	26.0	50.00	C1	Control	R1
5	5	37.0	100.00	C1	Control	R1
6	6	32.0	200.00	C1	Control	R1

Not Assuming the First Row in the CSV File is Labels

- The column labels will be “V1”, “V2”, etc…

[2]:
```
# Getting data from file "rabbit.csv"
rabbit_sample <- read.csv("datasets/rabbit.csv", header = FALSE)

# Printing first few lines of the dataframe
head(rabbit_sample)
```

A data.frame: 6 × 6

	V1 <int>	V2 <chr>	V3 <chr>	V4 <chr>	V5 <chr>	V6 <chr>
1	NA	BPchange	Dose	Run	Treatment	Animal
2	1	0.5	6.25	C1	Control	R1
3	2	4.5	12.5	C1	Control	R1
4	3	10	25	C1	Control	R1
5	4	26	50	C1	Control	R1
6	5	37	100	C1	Control	R1

Using Custom Column Names

- The rule is the same as rows.

[3]:
```
# Getting data from file "rabbit.csv"
rabbit_sample <- read.csv("datasets/rabbit.csv", col.names = c("A", "B", "C",␣
 ↪"D", "E", "F"))

# Printing first few lines of the dataframe
head(rabbit_sample)
```

A data.frame: 6 × 6

	A <int>	B <dbl>	C <dbl>	D <chr>	E <chr>	F <chr>
1	1	0.5	6.25	C1	Control	R1
2	2	4.5	12.50	C1	Control	R1
3	3	10.0	25.00	C1	Control	R1
4	4	26.0	50.00	C1	Control	R1
5	5	37.0	100.00	C1	Control	R1
6	6	32.0	200.00	C1	Control	R1

2.1.2 Write CSV Files

The Syntax

```
write.csv(x, file = "", quote = TRUE, eol = "\n", na = "NA", row.names = TRUE, fileEncoding = "
```

- A more general implementation is `write.table`. Check `?write.table` for more detail.

Simple Use of `write.csv`

[4]:
```
# Write the data.frame to "testing.csv"
write.csv(rabbit_sample, "datasets/testing.csv")
```

The file "testing.csv" contains:

```
"","A","B","C","D","E","F"
"1",1,0.5,6.25,"C1","Control","R1"
"2",2,4.5,12.5,"C1","Control","R1"
"3",3,10,25,"C1","Control","R1"
"4",4,26,50,"C1","Control","R1"
"5",5,37,100,"C1","Control","R1"
"6",6,32,200,"C1","Control","R1"
```

2.2 XLSX Files

2.2.1 Read XLSX Files

The Syntax

```
read.xlsx(
       file,
       sheetIndex,
       sheetName = NULL,
       rowIndex = NULL,
```

```
        startRow = NULL,
        endRow = NULL,
        colIndex = NULL,
        as.data.frame = TRUE,
        header = TRUE,
        colClasses = NA,
        keepFormulas = FALSE,
        encoding = "unknown",
        password = NULL,
        ...
)
```

Here … are other arguments to 'data.frame', for example 'stringsAsFactors'

Read a CSV File with Default Options

- `xlsx::read.xlsx` returns a data.frame
- The xlsx file used for demonstration contains the following data:

	A	B	C	D	E	F
1		Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
2	1	5.1	3.5	1.4	0.2	setosa
3	2	4.9	3	1.4	0.2	setosa
4	3	4.7	3.2	1.3	0.2	setosa
5	4	4.6	3.1	1.5	0.2	setosa
6	5	5	3.6	1.4	0.2	setosa
7	6	5.4	3.9	1.7	0.4	setosa
8	7	4.6	3.4	1.4	0.3	setosa

[5]:
```
# Loading the xlsx library
library(xlsx)

# Get the iris dataset from iris.xlsx, the second argument is the index of the␣
 ↪worksheet in the xlsx file.
iris_table <- xlsx::read.xlsx("datasets/iris.xlsx", 1)

# Print first few lines of the table
head(iris_table)
```

A data.frame: 6 × 6

	NA. <chr>	Sepal.Length <dbl>	Sepal.Width <dbl>	Petal.Length <dbl>	Petal.Width <dbl>	Species <chr>
1	1	5.1	3.5	1.4	0.2	setosa
2	2	4.9	3.0	1.4	0.2	setosa
3	3	4.7	3.2	1.3	0.2	setosa
4	4	4.6	3.1	1.5	0.2	setosa
5	5	5.0	3.6	1.4	0.2	setosa
6	6	5.4	3.9	1.7	0.4	setosa

2.2.2 Write XLSX Files

The Syntax

```
write.xlsx(
       x,
       file,
       sheetName = "Sheet1",
       col.names = TRUE,
       row.names = TRUE,
       append = FALSE,
       showNA = TRUE,
       password = NULL
     )
```

Write a CSV File with Default Options

[6]:
```
# Staff table to export
staff_table = data.frame(
    ID = c(1L, 2L, 3L, 4L),
    Name = c("Tom", "Ann", "Peter", "Kelly"),
    Phone = c(73490245L, 77990904L, 47876737L, 35146136L)
)

# Write the xlsx file to the file namely staff_table.xlsx
xlsx::write.xlsx(staff_table, "datasets/staff_table.xlsx", append = FALSE)
```

- The output xlsx file:

	A	B	C	D	E
1		ID	Name	Phone	
2	1	1	Tom	73490245	
3	2	2	Ann	77990904	
4	3	3	Peter	47876737	
5	4	4	Kelly	35146136	
6					
7					

3 Database

3.1 MySQL

3.1.1 Connect to MySQL Server

To connection to MySQL servers, we need to include two libraries:

[7]:
```
# Include libraries for MySQL connection
library(DBI)
library(RMySQL)
```

Then we connect to database namely "classicmodels" on the MySQL server at 127.0.0.1 using function DBI::dbConnect:

[8]:
```
# Create a connection object and store it in "con"
con <- DBI::dbConnect(RMySQL::MySQL(),          # The driver to communicate␣
 ↪with the server
                      dbname="classicmodels",   # The name of the database to␣
 ↪access on the server
                      host="127.0.0.1",         # The ip / URL / hostname of␣
 ↪the server
                      user="alan",              # user name to login
                      password="password")      # password for the user ID
```

Now the connection pipe is stored in object `con`. To list tables, we could use `DBI::dbListTables`.

[9]:
```
# Get the list of table in the database
DBI::dbListTables(conn = con)
```

1. 'NameList' 2. 'customers' 3. 'employees' 4. 'offices' 5. 'orderdetails' 6. 'orders' 7. 'payments' 8. 'productlines' 9. 'products'

3.1.2 Reading Tables by SQL Commands

Using `DBI::dbGetQuery`

- Syntax:

`dbGetQuery(conn, statement, ...)`

- `DBI::dbGetQuery` returns a data.frame.

[10]:
```
# Get the data.frame from the database based on the SQL statement
select_result <- DBI::dbGetQuery(conn = con, statement = "
    select customerNumber,customerName,phone from customers;
")

# Print out the class of the object select_result
cat("\nThe type of the output object:", class(select_result) ,". \n")

# Print first few lines of the object select_result
head(select_result)
```

```
The type of the output object: data.frame .
```

A data.frame: 6 × 3

	customerNumber <int>	customerName <chr>	phone <chr>
1	103	Atelier graphique	40.32.2555
2	112	Signal Gift Stores	7025551838
3	114	Australian Collectors, Co.	03 9520 4555
4	119	La Rochelle Gifts	40.67.8555
5	121	Baane Mini Imports	07-98 9555
6	124	Mini Gifts Distributors Ltd.	4155551450

Using `DBI::dbSendQuery` and `DBI::dbFetch`

- Syntax:

`dbSendQuery(conn, statement, ...)`

- `DBI::dbSendQuery` returns a S4 object. The S4 object can be translate to data.frame by `DBI::dbFetch`.
- The syntax of `DBI::dbFetch` :

  `dbFetch(res, n = -1, ...)`

  - Here n is the number of records to retrieve.

[11]:
```
# Get the S4 object from the database based on the SQL statement
select_result_raw <- DBI::dbSendQuery(conn = con, statement = "
    select customerNumber,customerName,state from customers;
")

# Translate the S4 object into data.frame
select_result <- DBI::dbFetch(select_result_raw)

# Print the class of the object select_result
cat("\nThe type of the output object:", class(select_result) ,". \n")

# Print first few lines of the object select_result
head(select_result)
```

`The type of the output object: data.frame .`

A data.frame: 6 × 3

	customerNumber <int>	customerName <chr>	state <chr>
1	103	Atelier graphique	NA
2	112	Signal Gift Stores	NV
3	114	Australian Collectors, Co.	Victoria
4	119	La Rochelle Gifts	NA
5	121	Baane Mini Imports	NA
6	124	Mini Gifts Distributors Ltd.	CA

3.1.3 Adding an Entry to a Table

- There are two routine to add entries to tables. But I found only `DBI::dbWriteTable` is working in the current scenario.
- `DBI::dbWriteTable` has a syntax:

  `dbWriteTable(conn, name, value, ...)`

  - `...` includes:
    1. 'row.names' (default: 'FALSE')
    2. 'overwrite' (default: 'FALSE')
    3. 'append' (default: 'FALSE')
    4. 'field.types' (default: 'NULL')
    5. 'temporary' (default: 'FALSE')

[12]:
```
# Create a data.frame for new entries
new_entry = data.frame(
    customerNumber = c(1001L,1002L),
    customerName = c("Tom", "Mary"),
    state = c("NA", "NY"),
    phone = c(173173173, 246246246)
)

# Show the content of the new entries
print(new_entry)

# Appending new rows in the table namely customers
DBI::dbWriteTable(conn = con, "customers", new_entry, append=TRUE, row.
 ↪names=FALSE)

# Print out the new entries to show their existence
select_result <- DBI::dbGetQuery(conn = con, statement = "
    select customerNumber,customerName,state,phone from customers where␣
 ↪customerNumber > 1000;
")

# Print first few lines of the object select_result
head(select_result)
```

```
  customerNumber customerName state     phone
1           1001          Tom    NA 173173173
2           1002         Mary    NY 246246246
```

TRUE

A data.frame: 2 × 4

	customerNumber <int>	customerName <chr>	state <chr>	phone <chr>
1	1001	Tom	NA	173173173
2	1002	Mary	NY	246246246

3.1.4 Deleting an Entry from a Table

[13]:
```
# Delete the entries by a SQL comment
result <- DBI::dbSendStatement(con, "delete from customers where customernumber␣
 ↪> 1000 ;")

# Attempted to select new records to show the deletion
select_result <- DBI::dbGetQuery(conn = con, statement = "
    select customerNumber,customerName,state,phone from customers where␣
 ↪customerNumber > 1000;
")

# Print first few lines of the object select_result
head(select_result)
```

A data.frame: 0 × 4

customerNumber	customerName	state	phone
<int>	<chr>	<chr>	<chr>

3.2 SQLite

4 Packages for Data Handling

[ ]:
```

```