02 Data Import and Manipulation

June 22, 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 <fct>	Treatment <fct>	Animal <fct>
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 <fct>	V3 <fct>	V4 <fct>	V5 <fct>	V6 <fct>
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 <fct>	E <fct>	F <fct>
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. <fct>	Sepal.Length <dbl>	Sepal.Width <dbl>	Petal.Length <dbl>	Petal.Width <dbl>	Species <fct>
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

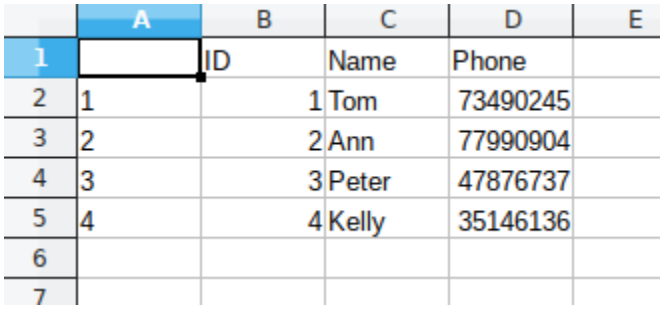
```
write.xlsx(
       x,
       file,
       sheetName = "Sheet1",
       col.names = TRUE,
       row.names = TRUE,
       append = FALSE,
       showNA = TRUE,
       password = NULL
     )
```

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. 'committees' 2. 'customers' 3. 'employees' 4. 'members' 5. 'offices' 6. 'orderdetails' 7. 'orders' 8. 'payments' 9. 'productlines' 10. 'products'

3.1.2 Reading Tables Through the DBI Interface

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

Getting the Whole Table

- If we want to retrieve the whole table, `DBI::dbReadTable` will be a shorter command.

[12]:
```
# Storing the Whole Table
whole_table <- DBI::dbReadTable(con, "offices")

# Show the first few lines of the data.frame whole_table
print(whole_table[1:3,])
```

```
  officeCode          city           phone         addressLine1 addressLine2
1          1 San Francisco +1 650 219 4782    100 Market Street    Suite 300
2          2        Boston +1 215 837 0825     1550 Court Place    Suite 102
3          3           NYC +1 212 555 3000 523 East 53rd Street      apt. 5A
  state country postalCode territory
1    CA     USA      94080        NA
2    MA     USA      02107        NA
3    NY     USA      10022        NA
```

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')

[13]:
```
# 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

- DBI seems not include a routine to delete an entry from tables. However SQL statement is still a working option.

[14]:
```
# 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 <int>	customerName <chr>	state <chr>	phone <chr>

3.1.5 Disconnect the Databse

[15]:
```
# The connection stored in con will be disconnected
DBI::dbDisconnect(con)
```

TRUE

3.2 SQLite

3.2.1 Open a SQLite File

- Like MySQL, this operation requires DBI library, while RSQLite is the driver pacakge to enable the connection.

[16]:
```
# Loading the required libraries
library(DBI, RSQLite)
```

- SQLite is server-less. The database is stored in a database file. Once the database file is connected, we may use it as if a SQL server.
- Like the MySQL example, we used `DBI::dbConnect` to open the sqlite file and store the connection object in `con`.
- There is an importent option called `flags`. This option controls the mode of database file opening.
  - If `flags=RSQLite::SQLITE_RWC` implies the database file is readable, writable, and creatable (if it does not exist).
  - If `flags=RSQLite::SQLITE_RO` implies the database file will be read-only in the follow operation.
  - `flags=RSQLite::SQLITE_RWC` is the default.

[17]:
```
# Open the sqlite file and store the connection object in con
con <- DBI::dbConnect(RSQLite::SQLite(), "datasets/patient_record.sqlite",␣
 ↪flags=RSQLite::SQLITE_RWC)
```

- Here we may query the list of tables inside the database.

[18]:
```
# Getting the list of table
DBI::dbListTables(con)
```

1. 'Hospitals' 2. 'PatientRecord' 3. 'Patients'

3.2.2 Reading Tables Through DBI Functions

- Like the examples in the MySQL section, DBI functions are workable in SQLite.

Using `DBI::dbGetQuery`

[19]:
```
# Get the PatientRecord from the database
patient_record_RH <- DBI::dbGetQuery(con, "select * from PatientRecord where␣
 ↪Hospital == \"RH\";")

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

A data.frame: 6 × 5

	Name <chr>	StartDate <chr>	EndDate <chr>	Hospital <chr>	Ward <chr>
1	Chantelle	2000-04-16	2000-04-21	RH	8C
2	Tonita	2000-09-16	2000-09-26	RH	4C
3	Ned	2000-10-21	2000-10-26	RH	6A
4	Silva	2000-10-25	2000-10-30	RH	9A
5	Johnnie	2001-02-10	2001-02-15	RH	7A
6	Chantelle	2001-03-29	2001-04-04	RH	6B

Using DBI::dbSendQuery and DBI::dbFetch

- In SQLite, object returned from `dbSendQuery` need to be cleaned by `dbClearResult` after the usage.

[20]:
```
# Get the PatientRecord from the database
patient_record_count_raw <- DBI::dbSendQuery(con, "select Name,count() from␣
↪PatientRecord group by Name;")

# Fetch the raw data to data.frame
patient_record_count <- DBI::dbFetch(patient_record_count_raw)

# Clean up the object created by DBI::dbSendQuery
DBI::dbClearResult(patient_record_count_raw)

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

A data.frame: 6 × 2

	Name <chr>	count() <int>
1	Alleen	5
2	Alona	2
3	Barb	4
4	Bridgett	2
5	Chantelle	3
6	Charla	3

Using `DBI::dbReadTable`

[21]:
```
# Storing the Whole Table
patient_record_all <- DBI::dbReadTable(con, "PatientRecord")

# Print first few records
print(patient_record_all[1:6,])
```

```
       Name  StartDate    EndDate Hospital Ward
1 Chantelle 2000-04-16 2000-04-21       RH   8C
2     Silva 2000-05-07 2000-05-16     TSKH   5A
3  Maybelle 2000-06-10 2000-06-13     WCHH   5A
```

```
4 Wilhemina 2000-06-12 2000-06-18     WCHH   5A
5    Alleen 2000-07-07 2000-07-17      SJH   6A
6   Natalia 2000-07-25 2000-08-02    PYNEH   8B
```

3.2.3 Adding an Entry to a Table

[22]:
```r
# Making a new entry to the table
new_entry <- data.frame(
    Name = c("Dummy"),
    StartDate = c("2020-01-25"),
    EndDate = c("2020-01-28"),
    Hospital = c("XXH"),
    Ward = c("10C")
)

# Append the entry to the table
DBI::dbWriteTable(con, "PatientRecord", new_entry, append = TRUE)

# Show that the newly appended entry exist
DBI::dbGetQuery(con, "select * from PatientRecord where Name == 'Dummy';")
```

A data.frame: 1 × 5

Name	StartDate	EndDate	Hospital	Ward
<chr>	<chr>	<chr>	<chr>	<chr>
Dummy	2020-01-25	2020-01-28	XXH	10C

3.2.4 Delete an Entry from a Table

- In SQLite, the function `DBI::dbSendStatement` will also return result concerning the SQL outcome. The SQL outcome should be cleaned after used, and before the next SQL statement.

[23]:
```r
# Running the SQL to Delete the just-append entry
DBI::dbClearResult(DBI::dbSendStatement(con, "delete from PatientRecord where␣
 ↪Name == 'Dummy';"))

# Show that the entry nolonger exist
DBI::dbGetQuery(con, "select * from PatientRecord where Name == 'Dummy';")
```

A data.frame: 0 × 5

Name	StartDate	EndDate	Hospital	Ward
<chr>	<chr>	<chr>	<chr>	<chr>

3.2.5 Disconnect the Database

[24]:
```r
# Disconnect using dbDisconnect
DBI::dbDisconnect(con)
```

4 Data Manipulation

Loading the SQLite Database for Demonstration.

[25]:
```
# Open the sqlite file and store the connection object in con
con <- DBI::dbConnect(RSQLite::SQLite(), "datasets/patient_record.sqlite",␣
 ↪flags=RSQLite::SQLITE_RO)
```

In the database, there are three tables:

[26]:
```
# List all tables
DBI::dbListTables(con)

# Getting three tables from the database
PatientRecord.Table <- DBI::dbReadTable(con, "PatientRecord")
Hospitals.Table <- DBI::dbReadTable(con, "Hospitals")
Patients.Tabble <- DBI::dbReadTable(con, "Patients")

# Print out first few lines of the tables
head(PatientRecord.Table)
head(Hospitals.Table)
head(Patients.Tabble)

# Disconnnect the database
DBI::dbDisconnect(con)
```

1. 'Hospitals' 2. 'PatientRecord' 3. 'Patients'

A data.frame: 6 × 5

	Name <chr>	StartDate <chr>	EndDate <chr>	Hospital <chr>	Ward <chr>
1	Chantelle	2000-04-16	2000-04-21	RH	8C
2	Silva	2000-05-07	2000-05-16	TSKH	5A
3	Maybelle	2000-06-10	2000-06-13	WCHH	5A
4	Wilhemina	2000-06-12	2000-06-18	WCHH	5A
5	Alleen	2000-07-07	2000-07-17	SJH	6A
6	Natalia	2000-07-25	2000-08-02	PYNEH	8B

A data.frame: 6 × 2

	Hospital <chr>	EquipGrade <chr>
1	CCH	Moderate
2	PYNEH	Low
3	RH	Low
4	SJH	Moderate
5	TSKH	Moderate
6	TWEH	Low

A data.frame: 6 × 3

	Name <chr>	Sex <chr>	HomePhone <int>
1	Yung	F	26370360
2	Lucas	M	21470543
3	Staci	F	21537227
4	Jesusita	M	29738952
5	Johnnie	F	20976943
6	Jadwiga	F	27701614

4.1 Filtering

4.1.1 Filtering / Selecting by Index / Indices

- Like other programming languages, square brackets accpets numbers to select rows and columns.

[27]:
```
# Print the first row and all columns of PatientRecord.Table
print(PatientRecord.Table[1,])
cat("\n")

# Print first two row and all columns of PatientRecord.Table
print(PatientRecord.Table[1:2,])
cat("\n")

# Print the first row and first two columns of PatientRecord.Table
print(PatientRecord.Table[1:2,1:2])
```

```
       Name  StartDate    EndDate Hospital Ward
1 Chantelle 2000-04-16 2000-04-21       RH   8C

       Name  StartDate    EndDate Hospital Ward
1 Chantelle 2000-04-16 2000-04-21       RH   8C
2     Silva 2000-05-07 2000-05-16     TSKH   5A

       Name  StartDate
1 Chantelle 2000-04-16
2     Silva 2000-05-07
```

4.1.2 Filtering by Criteria

- In R, square bracket accepts also boolean vector to filter data.

[28]:
```
# The returned table will contain rows being TRUE to the logical statement
head(Hospitals.Table[Hospitals.Table$EquipGrade %in% c("High", "Moderate"), ])
cat("\n")

# Combining two logical statement is acceptable
```

```
head(PatientRecord.Table[PatientRecord.Table$Ward == "8C" &
                         PatientRecord.Table$Hospital == "RH" ,])
cat("\n")
```

A data.frame: 4 × 2

	Hospital <chr>	EquipGrade <chr>
1	CCH	Moderate
4	SJH	Moderate
5	TSKH	Moderate
7	WCHH	High

A data.frame: 2 × 5

	Name <chr>	StartDate <chr>	EndDate <chr>	Hospital <chr>	Ward <chr>
1	Chantelle	2000-04-16	2000-04-21	RH	8C
43	Sunday	2001-07-14	2001-07-20	RH	8C

4.2 Sorting

- The function `order` returns the rank of the vector in its argument.
- By putting function `order`, we may sort the data frame.

4.2.1 Sorting with a Ascending Order

- The function `order` enables an ascending order by default.

[29]:
```
# Function order returns the rank of vector entries
print(order(PatientRecord.Table$Name))

# By using the function order, the table is sorted along the Name column
PatientRecord.Table.Sorted <- PatientRecord.Table[order(PatientRecord.
 ↪Table$Name),]

# Print the first 10 lines of the sorted table
head(PatientRecord.Table.Sorted, 10)
```

```
  [1]   5  33  67 129 140  50  87  17  72  83 122  61  79   1  26 106  20  90
 [19] 114  34 110 135 151 157  30  63 119  48 113 143 149 156  32  78  29  93
 [37]  22  39  59  73 121 131 138  53 115  15  45  13  66  58  69  95 124  38
 [55] 109 139 148 153  62  74  23  56  19  88  16  47  42  84  51  71  25  70
 [73] 123 142  36  60 100 120   7  52 117  28 105  64  82 104  46  80 108 136
 [91] 141  40  99  27 103   3  75  91 111   6  54 116 144 152   9  21 102 132
[109] 134  31  97 125 147  37  86 118 137  41 112  12  89 128  49  76  92 107
[127] 127  24  77  44  81   2  10  11  94 126 145 154  43  68  57  85 133 150
[145] 155   8  55  98 130 146   4  65 101  18  96  14  35
```

A data.frame: 10 × 5

	Name <chr>	StartDate <chr>	EndDate <chr>	Hospital <chr>	Ward <chr>
5	Alleen	2000-07-07	2000-07-17	SJH	6A
33	Alleen	2001-05-27	2001-06-05	CCH	4A
67	Alleen	2002-03-06	2002-03-14	TSKH	6B
129	Alleen	2004-01-08	2004-01-11	TWEH	8B
140	Alleen	2005-01-08	2005-01-16	WCHH	9C
50	Alona	2001-08-19	2001-08-28	RH	9C
87	Alona	2002-08-14	2002-08-17	TWEH	3A
17	Barb	2001-01-28	2001-01-31	TWEH	6C
72	Barb	2002-04-03	2002-04-11	SJH	8A
83	Barb	2002-07-27	2002-07-30	RH	9C

4.2.2 Sorting with a Descending Order

- `order` has an option `decreasing`. By assigning a boolean value, the direction of sorting can be controlled.

[30]:
```
# The descending order
PatientRecord.Table.Sorted <- PatientRecord.Table[order(PatientRecord.
↪Table$Name, decreasing = TRUE),]

# Print the first 10 lines of the sorted table
head(PatientRecord.Table.Sorted, 10)
```

A data.frame: 10 × 5

	Name <chr>	StartDate <chr>	EndDate <chr>	Hospital <chr>	Ward <chr>
14	Yung	2001-01-23	2001-02-01	SJH	4C
35	Yung	2001-06-22	2001-06-29	WCHH	4B
18	Xiomara	2001-02-09	2001-02-16	WCHH	8B
96	Xiomara	2002-12-02	2002-12-12	TSKH	5C
4	Wilhemina	2000-06-12	2000-06-18	WCHH	5A
65	Wilhemina	2001-12-04	2001-12-10	TSKH	9C
101	Wilhemina	2002-12-31	2003-01-05	PYNEH	3A
8	Tonita	2000-09-16	2000-09-26	RH	4C
55	Tonita	2001-10-01	2001-10-06	TSKH	7B
98	Tonita	2002-12-18	2002-12-24	WCHH	9B

4.2.3 Sorting along Multiple Columns

- Function `order` can accept multiple columns.

[31]:
```
# The descending order in Name and StartDate
PatientRecord.Table.Sorted <- PatientRecord.Table[
    order(PatientRecord.Table$Name,
          PatientRecord.Table$StartDate,
          decreasing = TRUE
```

```
),]

# Print the first 10 lines of the sorted table
head(PatientRecord.Table.Sorted, 10)
```

A data.frame: 10 × 5

	Name <chr>	StartDate <chr>	EndDate <chr>	Hospital <chr>	Ward <chr>
35	Yung	2001-06-22	2001-06-29	WCHH	4B
14	Yung	2001-01-23	2001-02-01	SJH	4C
96	Xiomara	2002-12-02	2002-12-12	TSKH	5C
18	Xiomara	2001-02-09	2001-02-16	WCHH	8B
101	Wilhemina	2002-12-31	2003-01-05	PYNEH	3A
65	Wilhemina	2001-12-04	2001-12-10	TSKH	9C
4	Wilhemina	2000-06-12	2000-06-18	WCHH	5A
146	Tonita	2005-07-31	2005-08-05	RH	3B
130	Tonita	2004-03-02	2004-03-09	RH	3B
98	Tonita	2002-12-18	2002-12-24	WCHH	9B

4.2.4 Sorting along Multiple Columns in Different Directions

- By make the concerning column a `factor`, we may use `as.numeric` to choose a reverse by accompanying a minus sign.

[32]:
```
# The ascending order in Name and descending order in StartDate
PatientRecord.Table.Sorted <- PatientRecord.Table[
    order(PatientRecord.Table$Name,
          -as.numeric(factor(PatientRecord.Table$StartDate)),
          decreasing = FALSE
),]

# Print the first 10 lines of the sorted table
head(PatientRecord.Table.Sorted, 10)
```

A data.frame: 10 × 5

	Name <chr>	StartDate <chr>	EndDate <chr>	Hospital <chr>	Ward <chr>
140	Alleen	2005-01-08	2005-01-16	WCHH	9C
129	Alleen	2004-01-08	2004-01-11	TWEH	8B
67	Alleen	2002-03-06	2002-03-14	TSKH	6B
33	Alleen	2001-05-27	2001-06-05	CCH	4A
5	Alleen	2000-07-07	2000-07-17	SJH	6A
87	Alona	2002-08-14	2002-08-17	TWEH	3A
50	Alona	2001-08-19	2001-08-28	RH	9C
122	Barb	2003-08-27	2003-09-06	WCHH	5A
83	Barb	2002-07-27	2002-07-30	RH	9C
72	Barb	2002-04-03	2002-04-11	SJH	8A

Reset Row-Names

- By assigning `NULL` to `rowname`, the rowname can be reset.

[33]:
```
# Reset the rowname
rownames(PatientRecord.Table.Sorted) <- NULL

# Print the first 10 lines of the sorted table
head(PatientRecord.Table.Sorted, 10)
```

A data.frame: 10 × 5

	Name <chr>	StartDate <chr>	EndDate <chr>	Hospital <chr>	Ward <chr>
1	Alleen	2005-01-08	2005-01-16	WCHH	9C
2	Alleen	2004-01-08	2004-01-11	TWEH	8B
3	Alleen	2002-03-06	2002-03-14	TSKH	6B
4	Alleen	2001-05-27	2001-06-05	CCH	4A
5	Alleen	2000-07-07	2000-07-17	SJH	6A
6	Alona	2002-08-14	2002-08-17	TWEH	3A
7	Alona	2001-08-19	2001-08-28	RH	9C
8	Barb	2003-08-27	2003-09-06	WCHH	5A
9	Barb	2002-07-27	2002-07-30	RH	9C
10	Barb	2002-04-03	2002-04-11	SJH	8A

4.3 Column Shifting with `dplyr`

- Library `dplyr` is using for data frame manipulation.
- Here we will `dplyr::lead` and `dplyr::lag`

[34]:
```
# Loading the library
library(dplyr, warn.conflicts = FALSE)

# For example, dplyr::lead can shift a vector backwards
v = 1:10
print(v)
print(dplyr::lead(v, 1))

# dplyr::lag can shift a vector forwards
print(dplyr::lag(v, 1))
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
 [1]  2  3  4  5  6  7  8  9 10 NA
 [1] NA  1  2  3  4  5  6  7  8  9
```

Example: Making Columns for Previous Hospitals and Previous Ward for Each Patient

[35]:
```
# Making a new data.frame to store PatientRecord.Table with Last hospital and␣
 ↪Last Ward
PatientRecord.Table.withLast <- PatientRecord.Table

# The for loop run over factors of PatientRecord.Table$Name
```

```
for (name_i in levels(factor(PatientRecord.Table$Name))){
    # Store vector of hospitals of patient name_i
    tmp <- PatientRecord.Table$Hospital[PatientRecord.Table$Name == name_i]
    # Shift the vector forwards and store it in the new column namely␣
 ↪LastHospital
    PatientRecord.Table.withLast$LastHospital[PatientRecord.Table$Name ==␣
 ↪name_i] <- dplyr::lag(tmp, 1)

    # Store vector of wards of patient name_i
    tmp <- PatientRecord.Table$Ward[PatientRecord.Table$Name == name_i]
    # Shift the vector forwards and store it in the new column namely LastWard ␣
 ↪
    PatientRecord.Table.withLast$LastWard[PatientRecord.Table$Name == name_i]␣
 ↪<- dplyr::lag(tmp, 1)
}

# Checking for the patient Chantelle
print(PatientRecord.Table.withLast[PatientRecord.Table$Name == "Chantelle",])

cat("\n")

# Checking for the patient Alleen
print(PatientRecord.Table.withLast[PatientRecord.Table$Name == "Alleen",])
```

```
         Name  StartDate    EndDate Hospital Ward LastHospital LastWard
1   Chantelle 2000-04-16 2000-04-21       RH   8C         <NA>     <NA>
26  Chantelle 2001-03-29 2001-04-04       RH   6B           RH       8C
106 Chantelle 2003-01-20 2003-01-27     WCHH   3A           RH       6B

      Name  StartDate    EndDate Hospital Ward LastHospital LastWard
5   Alleen 2000-07-07 2000-07-17      SJH   6A         <NA>     <NA>
33  Alleen 2001-05-27 2001-06-05      CCH   4A          SJH       6A
67  Alleen 2002-03-06 2002-03-14     TSKH   6B          CCH       4A
129 Alleen 2004-01-08 2004-01-11     TWEH   8B         TSKH       6B
140 Alleen 2005-01-08 2005-01-16     WCHH   9C         TWEH       8B
```

4.4 Table Joining with `dplyr`

- `dplyr` provides functions `inner_join`, `left_join`, `right_join`, `full_join` etc. The meanings of those functions are the same as their counterparts in SQL.
- For detail, please check https://dplyr.tidyverse.org/reference/join.html

Example: Inner-Join Two Tables

[36]:
```
# Print first few lines the outcome of inner-join of two tables
```

```
head(dplyr::inner_join(PatientRecord.Table, Hospitals.Table, by = c("Hospital"␣
↪= "Hospital")), 10)
```

A data.frame: 10 × 6

	Name <chr>	StartDate <chr>	EndDate <chr>	Hospital <chr>	Ward <chr>	EquipGrade <chr>
1	Chantelle	2000-04-16	2000-04-21	RH	8C	Low
2	Silva	2000-05-07	2000-05-16	TSKH	5A	Moderate
3	Maybelle	2000-06-10	2000-06-13	WCHH	5A	High
4	Wilhemina	2000-06-12	2000-06-18	WCHH	5A	High
5	Alleen	2000-07-07	2000-07-17	SJH	6A	Moderate
6	Natalia	2000-07-25	2000-08-02	PYNEH	8B	Low
7	Lawanda	2000-09-04	2000-09-14	WCHH	9A	High
8	Tonita	2000-09-16	2000-09-26	RH	4C	Low
9	Ned	2000-10-21	2000-10-26	RH	6A	Low
10	Silva	2000-10-25	2000-10-30	RH	9A	Low

Example: Left-Join Two Tables

[37]:
```
# Make an incomplete table to show left-join
Hospitals.Table.incomplete <- Hospitals.Table[1:2,]

# Print first few lines the outcome of left-join of two tables
head(dplyr::left_join(PatientRecord.Table, Hospitals.Table.incomplete, by =␣
↪c("Hospital" = "Hospital")), 10)
```

A data.frame: 10 × 6

	Name <chr>	StartDate <chr>	EndDate <chr>	Hospital <chr>	Ward <chr>	EquipGrade <chr>
1	Chantelle	2000-04-16	2000-04-21	RH	8C	NA
2	Silva	2000-05-07	2000-05-16	TSKH	5A	NA
3	Maybelle	2000-06-10	2000-06-13	WCHH	5A	NA
4	Wilhemina	2000-06-12	2000-06-18	WCHH	5A	NA
5	Alleen	2000-07-07	2000-07-17	SJH	6A	NA
6	Natalia	2000-07-25	2000-08-02	PYNEH	8B	Low
7	Lawanda	2000-09-04	2000-09-14	WCHH	9A	NA
8	Tonita	2000-09-16	2000-09-26	RH	4C	NA
9	Ned	2000-10-21	2000-10-26	RH	6A	NA
10	Silva	2000-10-25	2000-10-30	RH	9A	NA

- Then many entries in EquipGrade become NA, as expected for left-join with an incomplete table.

4.5 Other Operations Avilable in `dplyr`

In this subsection, toy datasets from MASS will be used for demonstractions.

4.5.1 Toy Datasets

Here we load the MASS library and use its Rabbit data frame.

[38]:
```
# Load the MASS library
library(MASS, warn.conflicts = FALSE)

# Assign the MASS::Rabbit data.frame to a new data.frame
rabbit <- MASS::Rabbit
```

Content of the data frame

[39]:
```
# Print first few lines
head(rabbit, 10)
```

A data.frame: 10 × 5

	BPchange <dbl>	Dose <dbl>	Run <fct>	Treatment <fct>	Animal <fct>
1	0.50	6.25	C1	Control	R1
2	4.50	12.50	C1	Control	R1
3	10.00	25.00	C1	Control	R1
4	26.00	50.00	C1	Control	R1
5	37.00	100.00	C1	Control	R1
6	32.00	200.00	C1	Control	R1
7	1.00	6.25	C2	Control	R2
8	1.25	12.50	C2	Control	R2
9	4.00	25.00	C2	Control	R2
10	12.00	50.00	C2	Control	R2

4.5.2 `dplyr::select()` Column Selction

[40]:
```
# Select columns of the data.frame
result <- rabbit %>% dplyr::select(BPchange, Dose, Animal)

# Print first few lines
head(result)
```

A data.frame: 6 × 3

	BPchange <dbl>	Dose <dbl>	Animal <fct>
1	0.5	6.25	R1
2	4.5	12.50	R1
3	10.0	25.00	R1
4	26.0	50.00	R1
5	37.0	100.00	R1
6	32.0	200.00	R1

4.5.3 `dplyr::rename` : Rename Column Name

[41]:
```
# Rename column namely BPchange
result <- rabbit %>% dplyr::rename(ChangeInBP = BPchange)

# Print first few lines
head(result,10)
```

A data.frame: 10 × 5

	ChangeInBP <dbl>	Dose <dbl>	Run <fct>	Treatment <fct>	Animal <fct>
1	0.50	6.25	C1	Control	R1
2	4.50	12.50	C1	Control	R1
3	10.00	25.00	C1	Control	R1
4	26.00	50.00	C1	Control	R1
5	37.00	100.00	C1	Control	R1
6	32.00	200.00	C1	Control	R1
7	1.00	6.25	C2	Control	R2
8	1.25	12.50	C2	Control	R2
9	4.00	25.00	C2	Control	R2
10	12.00	50.00	C2	Control	R2

4.5.4 `dplyr::mutate` : Create, Modify, and Delete Columns

Create a Column

[42]:
```
# Create a new column namely PBchangePlusDose by summing PBchange and Dose
result <- rabbit %>% dplyr::mutate(PBchangePlusDose = BPchange + Dose)

# Print first few lines
head(result,10)
```

A data.frame: 10 × 6

	BPchange <dbl>	Dose <dbl>	Run <fct>	Treatment <fct>	Animal <fct>	PBchangePlusDose <dbl>
1	0.50	6.25	C1	Control	R1	6.75
2	4.50	12.50	C1	Control	R1	17.00
3	10.00	25.00	C1	Control	R1	35.00
4	26.00	50.00	C1	Control	R1	76.00
5	37.00	100.00	C1	Control	R1	137.00
6	32.00	200.00	C1	Control	R1	232.00
7	1.00	6.25	C2	Control	R2	7.25
8	1.25	12.50	C2	Control	R2	13.75
9	4.00	25.00	C2	Control	R2	29.00
10	12.00	50.00	C2	Control	R2	62.00

Modify a Column

[43]:
```
# Modify a column by a formula
result <- rabbit %>% dplyr::mutate(BPchange = BPchange + Dose)
```

```
# Print first few lines
head(result,10)
```

A data.frame: 10 × 5

	BPchange <dbl>	Dose <dbl>	Run <fct>	Treatment <fct>	Animal <fct>
1	6.75	6.25	C1	Control	R1
2	17.00	12.50	C1	Control	R1
3	35.00	25.00	C1	Control	R1
4	76.00	50.00	C1	Control	R1
5	137.00	100.00	C1	Control	R1
6	232.00	200.00	C1	Control	R1
7	7.25	6.25	C2	Control	R2
8	13.75	12.50	C2	Control	R2
9	29.00	25.00	C2	Control	R2
10	62.00	50.00	C2	Control	R2

Delete a Column

[44]:
```
# Delete the BPchange column
result <- rabbit %>% dplyr::mutate(BPchange = NULL)

# Print first few lines
head(result,10)
```

A data.frame: 10 × 4

	Dose <dbl>	Run <fct>	Treatment <fct>	Animal <fct>
1	6.25	C1	Control	R1
2	12.50	C1	Control	R1
3	25.00	C1	Control	R1
4	50.00	C1	Control	R1
5	100.00	C1	Control	R1
6	200.00	C1	Control	R1
7	6.25	C2	Control	R2
8	12.50	C2	Control	R2
9	25.00	C2	Control	R2
10	50.00	C2	Control	R2

Special Example: Adding LastWard and LastHospital in the Table

[45]:
```
result <- PatientRecord.Table %>%
dplyr::group_by(Name) %>%
dplyr::mutate(LastHospital = lag(Hospital, 1)) %>%
dplyr::mutate(LastWard = lag(Ward, 1))

print(result[result$Name == "Chantelle",])

cat("\n")
```

```
print(result[result$Name == "Alleen",])
```

```
# A tibble: 3 x 7
# Groups:   Name [1]
  Name      StartDate  EndDate    Hospital Ward  LastHospital LastWard
  <chr>     <chr>      <chr>      <chr>    <chr> <chr>        <chr>
1 Chantelle 2000-04-16 2000-04-21 RH       8C    NA           NA
2 Chantelle 2001-03-29 2001-04-04 RH       6B    RH           8C
3 Chantelle 2003-01-20 2003-01-27 WCHH     3A    RH           6B

# A tibble: 5 x 7
# Groups:   Name [1]
  Name   StartDate  EndDate    Hospital Ward  LastHospital LastWard
  <chr>  <chr>      <chr>      <chr>    <chr> <chr>        <chr>
1 Alleen 2000-07-07 2000-07-17 SJH      6A    NA           NA
2 Alleen 2001-05-27 2001-06-05 CCH      4A    SJH          6A
3 Alleen 2002-03-06 2002-03-14 TSKH     6B    CCH          4A
4 Alleen 2004-01-08 2004-01-11 TWEH     8B    TSKH         6B
5 Alleen 2005-01-08 2005-01-16 WCHH     9C    TWEH         8B
```

4.5.5 `dplyr::arrange` : Arrange Rows by Column Values

[46]:
```
# Sorting the Animal in the ascending direction and Run in the Descending␣
 ↪Direction
result <- rabbit %>% dplyr::arrange(Animal, desc(Run))

# Print a subset of the table
result[result$Dose < 10.0,]
```

A data.frame: 10 × 5

	BPchange <dbl>	Dose <dbl>	Run <fct>	Treatment <fct>	Animal <fct>
1	1.25	6.25	M1	MDL	R1
7	0.50	6.25	C1	Control	R1
13	1.40	6.25	M2	MDL	R2
19	1.00	6.25	C2	Control	R2
25	0.75	6.25	M3	MDL	R3
31	0.75	6.25	C3	Control	R3
37	2.60	6.25	M4	MDL	R4
43	1.25	6.25	C4	Control	R4
49	2.40	6.25	M5	MDL	R5
55	1.50	6.25	C5	Control	R5

5 dbplyr and Database

- There is a `dbplyr` make `dplyr` functions workable on databases.
- We need not to load `dbplyr` separately.

Load Libraries

[47]:
```
library(DBI, RSQLite, dplyr, warn.conflicts = FALSE)
```

5.1 Directing Tables in Database to dplyr tbl objects

[48]:
```
# Open the sqlite file and store the connection object in con
con <- DBI::dbConnect(RSQLite::SQLite(), "datasets/patient_record.sqlite",␣
 ↪flags=RSQLite::SQLITE_RO)

# Point table PatientRecord to dplyr tbl object PatientRecord.tbl and print it
PatientRecord.tbl <- dplyr::tbl(con, "PatientRecord")
print(PatientRecord.tbl)

cat("\n")

# Point table Hospitals to dplyr tbl object Hospitals.tbl and print it
Hospitals.tbl <- dplyr::tbl(con, "Hospitals")
print(Hospitals.tbl)
```

```
# Source:   table<PatientRecord> [?? x 5]
# Database: sqlite 3.30.1
#   [/home/alan/lab/playground/R_self_teaching_notes/self_teaching_notes_R/datas
ets/patient_record.sqlite]
   Name      StartDate  EndDate    Hospital Ward
   <chr>     <chr>
```

```
<chr>      <chr>
<chr>
 1 Chantelle 2000-04-16 2000-04-21 RH       8C
 2 Silva     2000-05-07 2000-05-16 TSKH     5A
 3 Maybelle  2000-06-10 2000-06-13 WCHH     5A
 4 Wilhemina 2000-06-12 2000-06-18 WCHH     5A
 5 Alleen    2000-07-07 2000-07-17 SJH      6A
 6 Natalia   2000-07-25 2000-08-02 PYNEH    8B
 7 Lawanda   2000-09-04 2000-09-14 WCHH     9A
 8 Tonita    2000-09-16 2000-09-26 RH       4C
 9 Ned       2000-10-21 2000-10-26 RH       6A
10 Silva     2000-10-25 2000-10-30 RH       9A
# … with more rows

# Source:   table<Hospitals> [?? x 2]
# Database: sqlite 3.30.1
#   [/home/alan/lab/playground/R_self_teaching_notes/self_teaching_notes_R/datas
ets/patient_record.sqlite]
  Hospital EquipGrade
  <chr>    <chr>
1 CCH      Moderate
2 PYNEH    Low
3 RH       Low
4 SJH      Moderate
5 TSKH     Moderate
6 TWEH     Low
7 WCHH     High
```

5.2 Using %>% Pipe to Perform SQL Operations

5.2.1 Select

[49]:
```
Selected <- PatientRecord.tbl %>% dplyr::select(Name, Hospital, StartDate)

print(Selected)
```

```
# Source:   lazy query [?? x 3]
# Database: sqlite 3.30.1
#   [/home/alan/lab/playground/R_self_teaching_notes/self_teaching_notes_R/datas
ets/patient_record.sqlite]
   Name      Hospital StartDate
   <chr>     <chr>
<chr>
 1 Chantelle RH       2000-04-16
 2 Silva     TSKH     2000-05-07
```

```
 3 Maybelle  WCHH     2000-06-10
 4 Wilhemina WCHH     2000-06-12
 5 Alleen    SJH      2000-07-07
 6 Natalia   PYNEH    2000-07-25
 7 Lawanda   WCHH     2000-09-04
 8 Tonita    RH       2000-09-16
 9 Ned       RH       2000-10-21
10 Silva     RH       2000-10-25
# … with more rows
```

The SQL Query Behind

[50]:
```
Selected %>% dplyr::show_query()
```

```
<SQL>
SELECT `Name`, `Hospital`, `StartDate`
FROM `PatientRecord`
```

5.2.2 Group and Counting

[51]:
```
Counted <- PatientRecord.tbl %>% dplyr::select(Name) %>% dplyr::group_by(Name)␣
↪%>% count(Name)

print(Counted)
```

```
# Source:   lazy query [?? x 2]
# Database: sqlite 3.30.1
#   [/home/alan/lab/playground/R_self_teaching_notes/self_teaching_notes_R/datas
ets/patient_record.sqlite]
   Name          n
   <chr>     <int>
 1 Alleen        5
 2 Alona         2
 3 Barb          4
 4 Bridgett      2
 5 Chantelle     3
 6 Charla        3
 7 Claretha      5
 8 Debby         3
 9 Deshawn       5
10 Dierdre       2
# … with more rows
```

The SQL Query Behind

[52]:
```
Counted %>% dplyr::show_query()
```

```
<SQL>
SELECT `Name`, COUNT() AS `n`
FROM (SELECT `Name`
FROM `PatientRecord`)
GROUP BY `Name`
```