ADVANCED DATA PROCESSING

IOTB_ex63.R

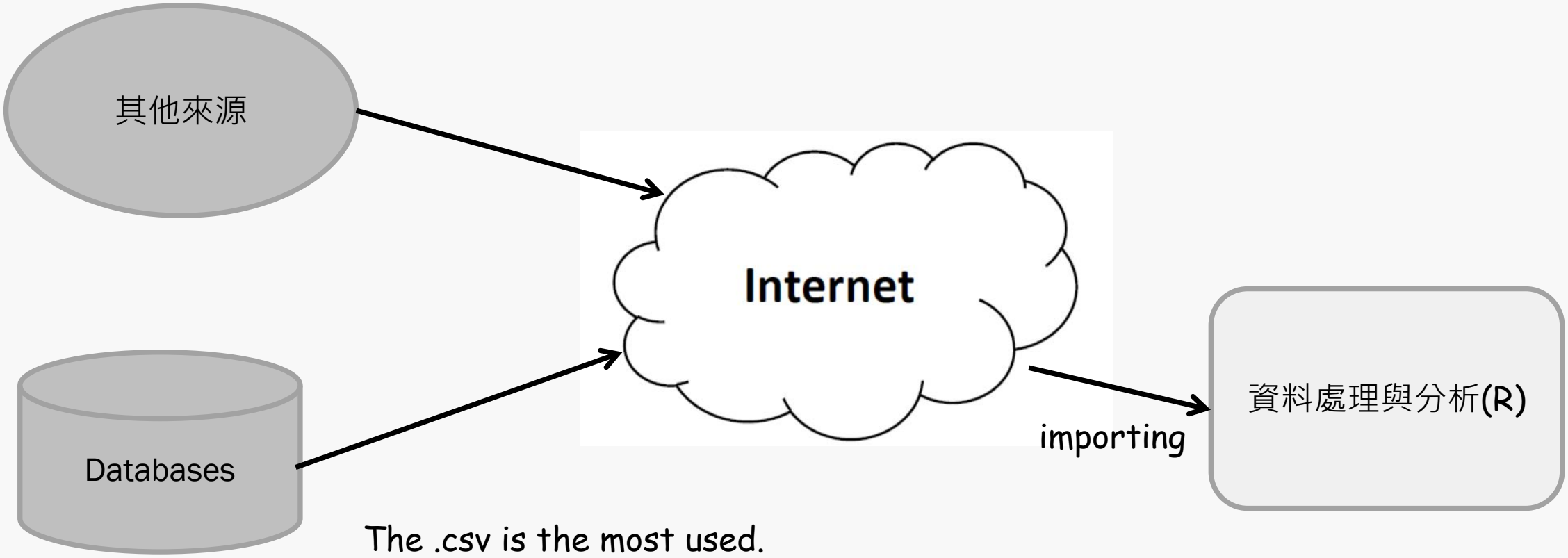
Contents

- Data Importing
- Data Reshaping

Data Importing(資料匯入)

Week 2

Process CSV Files



R - CSV Files

- In R, we can read data from files stored outside the R environment. We can also write data into files which will be stored and accessed by the operating system. R can read and write into various file formats like csv, excel, xml etc.
- In this session, we will learn to read data from a csv file and then write data into a csv file. The file should be present in current working directory so that R can read it. Of course we can also set our own directory and read files from there.

Input as CSV File

- The csv file is a text file in which the values in the columns are separated by a comma. Let's consider the following data present in the file named **csv-file.csv**.
- You can create this file using windows notepad by copying and pasting this data. Save the file as **csv-file.csv** using the save As All files(*.*) option in notepad.

```
id,name,salary,start_date,dept
1,Rick,623.3,2012-01-01,IT
2,Dan,515.2,2013-09-23,Operations
3,Michelle,611,2014-11-15,IT
4,Ryan,729,2014-05-11,HR
5,Gary,843.25,2015-03-27,Finance
6,Nina,578,2013-05-21,IT
7,Simon,632.8,2013-07-30,Operations
8,Guru,722.5,2014-06-17,Finance
```

Input as a CSV File

Following is a simple example of **read.csv()** function to read a CSV file available in your current working directory

```
> data <- read.csv("csv-file.csv")
> print(data)
  id      name salary start_date       dept
1  1      Rick 623.30   2012/1/1         IT
2  2       Dan 515.20  2013/9/23 Operations
3  3 Michelle 611.00 2014/11/15         IT
4  4      Ryan 729.00  2014/5/11         HR
5  5      Gary 843.25  2015/3/27    Finance
6  6      Nina 578.00  2013/5/21         IT
7  7     Simon 632.80  2013/7/30 Operations
8  8      Guru 722.50  2014/6/17    Finance
```

```
> data <- read.csv("csv-file.csv")
> print(is.data.frame(data))
[1] TRUE
> print(ncol(data))
[1] 5
> print(nrow(data))
[1] 8
```

csv-file.csv

```
id,name,salary,start_date,dept
1,Rick,623.3,2012/1/1,IT
2,Dan,515.2,2013/9/23,Operations
3,Michelle,611,2014/11/15,IT
4,Ryan,729,2014/5/11,HR
5,Gary,843.25,2015/3/27,Finance
6,Nina,578,2013/5/21,IT
7,Simon,632.8,2013/7/30,Operations
8,Guru,722.5,2014/6/17,Finance
```

Get the maximum salary

- Get the maximum salary from the **csv-file.csv**

  you created just now.

```
data <- read.csv("csv-file.csv")
sal <- max(data$salary) # Get the max salary from data frame.
print(sal)
```

```
[1] 843.25
```

- We can fetch rows meeting specific filter criteria by using **subset()** similar to an SQL where clause.

Subsetting Vectors, Matrices and Data Frames

Description

Return subsets of vectors, matrices or data frames which meet conditions.

Usage

```
subset(x, ...)

## Default S3 method:
subset(x, subset, ...)
```

Exercise

- Actually, we can fetch rows meeting specific filter criteria similar to a SQL where clause by using subset().

```
> print(data)
  id     name salary start_date       dept
1  1     Rick 623.30   2012/1/1         IT
2  2      Dan 515.20  2013/9/23 Operations
3  3 Michelle 611.00 2014/11/15         IT
4  4     Ryan 729.00  2014/5/11         HR
5  5     Gary 843.25  2015/3/27    Finance
6  6     Nina 578.00  2013/5/21         IT
7  7    Simon 632.80  2013/7/30 Operations
8  8     Guru 722.50  2014/6/17    Finance
```

(1)

```
id name salary start_date    dept
 5 Gary 843.25  2015/3/27 Finance
```

(2)

```
id      name salary start_date dept
 1      Rick  623.3   2012/1/1   IT
 3 Michelle  611.0 2014/11/15   IT
 6      Nina  578.0  2013/5/21   IT
```

(3)

```
id      name salary start_date dept
 1      Rick  623.3   2012/1/1   IT
 3 Michelle  611.0 2014/11/15   IT
```

(4)

```
id      name salary start_date    dept
 3 Michelle 611.00 2014/11/15      IT
 4      Ryan 729.00  2014/5/11      HR
 5      Gary 843.25  2015/3/27 Finance
 8      Guru 722.50  2014/6/17 Finance
```

Exercise-Writing into a CSV File

- R can create csv file form existing data frame. The **write.csv()** function is used to create the csv file. This file gets created in the working directory.
- You need to use subset() to filter the data from **csv-file.csv**, then output it into a file named as **output.csv** by using **write.csv()**.

csv-file.csv

```
id,name,salary,start_date,dept
1,Rick,623.3,2012/1/1,IT
2,Dan,515.2,2013/9/23,Operations
3,Michelle,611,2014/11/15,IT
4,Ryan,729,2014/5/11,HR
5,Gary,843.25,2015/3/27,Finance
6,Nina,578,2013/5/21,IT
7,Simon,632.8,2013/7/30,Operations
8,Guru,722.5,2014/6/17,Finance
```

過濾 csv-file.csv
選出 日期大於
2014.1.1

In output.csv

ex34.R | output.csv

```
"","id","name","salary","start_date","dept"
"3",3,"Michelle",611,"2014/11/15","IT"
"4",4,"Ryan",729,"2014/5/11","HR"
"5",5,"Gary",843.25,"2015/3/27","Finance"
"8",8,"Guru",722.5,"2014/6/17","Finance"
```

In R Studio

	X	id	name	salary	start_date	dept
1	3	3	Michelle	611.00	2014/11/15	IT
2	4	4	Ryan	729.00	2014/5/11	HR
3	5	5	Gary	843.25	2015/3/27	Finance
4	8	8	Guru	722.50	2014/6/17	Finance

由於Excel預設的CSV檔編碼是Big-5，而R輸出的CSV檔編碼是UTF-8，因此用Excel打開剛輸出的CSV檔，可能會是亂碼

R - JSON Files

- JSON file stores data as text in human-readable format. Json stands for JavaScript Object Notation. R can read JSON files using the rjson package.
- JSON 是個以純文字為基底去儲存和傳送簡單結構資料，你可以透過特定的格式去儲存任何資料(字串,數字,陣列,物件)，也可以透過物件或陣列來傳送較複雜的資料。一旦建立了您的 JSON 資料，就可以非常簡單的跟其他程式溝通或交換資料，因為 JSON 就只是純文字個格式
- JSON 的優點如下:
  - 相容性高
  - 格式容易瞭解，閱讀及修改方便
  - 支援許多資料格式 (number, string, booleans, nulls,array, associative array)
  - 許多程式都支援函式庫讀取或修改 JSON 資料

Install RJSON Package

- In the R console, you can issue the following command to install the rjson package.

```
install.packages("jsonlite")
any(grepl("jsonlite",installed.packages()))
library(jsonlite)
```

- jsonlite 是 R 的一個 JSON 格式資料處理套件. JSON（JavaScript Object Notation）是一種輕量級的資料交換格式，屬於 JavaScript 語言的子集，實作上相當容易，在網路上許多資料都會使用 JSON 的格式來傳遞。

使用mtcars

- 內建 **mtcars**: The data set was extracted from the 1974 *Motor Trend* US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).

```
> mtcars
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic       30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla    33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona     21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Dodge Challenger  15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
```

- **mtcars** 是 dataframe格式, fromJSON轉換完也是 dataframe格式

```
>   # 將 mtcars 這個 data frame 轉為 JSON 資料格式
> my.json <- toJSON(mtcars)
>   # 將 JSON 資料格式轉為 data frame
> my.df <- fromJSON(my.json)
>   # 檢查轉換後的資料是否一致
> all.equal(mtcars, my.df)
[1] TRUE
```

外部檔案導入處理

```
> write(my.json, "output.json")
> fromAJsonFile <- fromJSON("output.json")
> head(fromAJsonFile)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 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 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

將 json 格式轉為 R 向量

```
> json <- '["Mario", "Peach", null, "Bowser"]'
> fromJSON(json)
[1] "Mario"  "Peach"  NA       "Bowser"
```

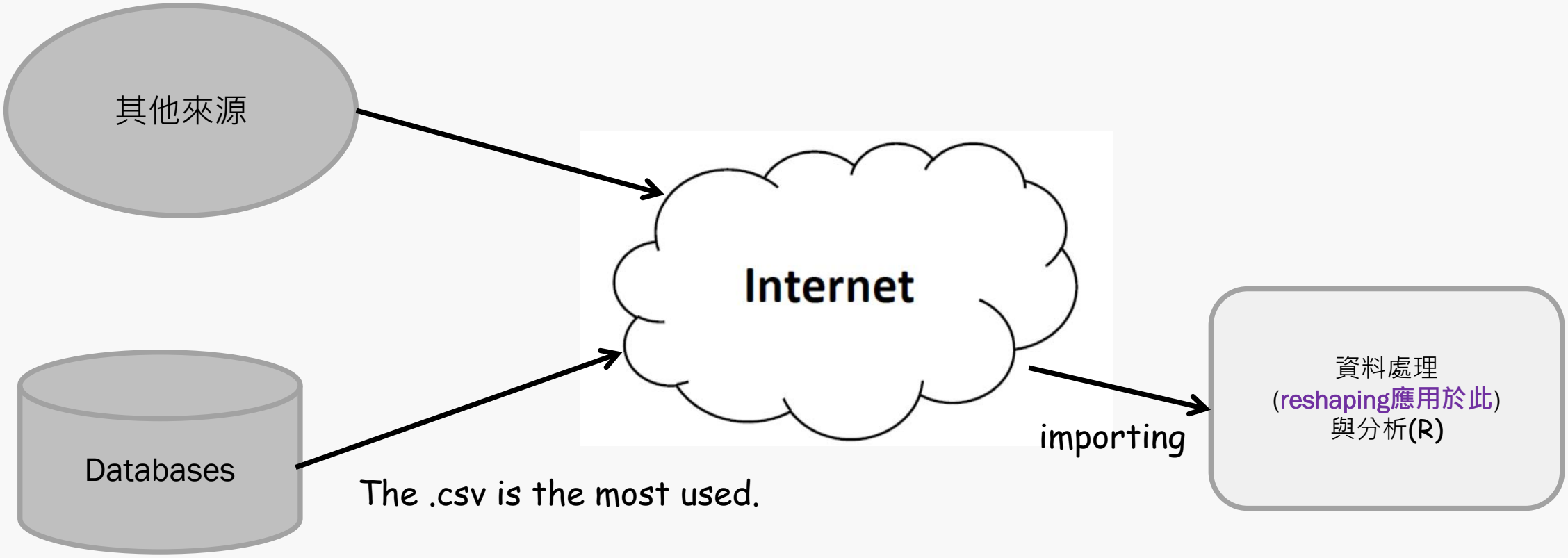
轉json為dataframe物件

```
> json <-
+   '[
+ {"Name" : "Mario", "Age" : 32, "Occupation" : "Plumber"},
+ {"Name" : "Peach", "Age" : 21, "Occupation" : "Princess"},
+ {},
+ {"Name" : "Bowser", "Occupation" : "Koopa"}
+ ]'
> (my.df <- fromJSON(json))
    Name Age Occupation
1  Mario  32    Plumber
2  Peach  21   Princess
3   <NA>  NA       <NA>
4 Bowser  NA      Koopa
```

```
> my.df #same as previous one
    Name Age Occupation
1  Mario  32    Plumber
2  Peach  21   Princess
3   <NA>  NA       <NA>
4 Bowser  NA      Koopa
```

Data Reshaping(資料重整)

Process CSV Files



Data Reshaping

- Data Reshaping in R is about changing the way data is organized into rows and columns.
- Most of the time data processing in R is done by taking the input data as a data frame. It is easy to extract data from the rows and columns of a data frame but there are situations when we need the data frame in a format that is different from format in which we received it.
- R has many functions to **split**, **merge** and **change** the rows to columns and vice-versa in a data frame.

Joining Columns and Rows in a Data Frame

- We can join multiple vectors to create a data frame using the **cbind()** function.
- Also we can merge two data frames using **rbind()** function.

Joining Columns and Rows in a Data Frame

- Example

```
city <- c("Tampa","Seattle","Hartford","Denver")
state <- c("FL","WA","CT","CO")
zipcode <- c(33602,98104,06161,80294)
addresses <- cbind(city,state,zipcode) # Combine above
cat("# # # # The First data frame\n") # Print a header.
print(addresses) # Print the data frame.

# Create another data frame with similar columns
new.address <- data.frame(
  city = c("Lowry","Charlotte"),
  state = c("CO","FL"),
  zipcode = c("80230","33949"),
  stringsAsFactors = FALSE
)
cat("# # # The Second data frame\n")
print(new.address)
all.addresses <- rbind(addresses,new.address)

cat("# # # The combined data frame\n")
print(all.addresses)
```

```
# # # # The First data frame
     city       state zipcode
[1,] "Tampa"    "FL"  "33602"
[2,] "Seattle"  "WA"  "98104"
[3,] "Hartford" "CT"   "6161"
[4,] "Denver"   "CO"  "80294"

# # # The Second data frame
       city       state   zipcode
1      Lowry      CO      80230
2      Charlotte  FL      33949

# # # The combined data frame
       city       state zipcode
1      Tampa      FL    33602
2      Seattle    WA    98104
3      Hartford   CT     6161
4      Denver     CO    80294
5      Lowry      CO    80230
6     Charlotte   FL    33949
```

Merging Data Frames

- We can merge two data frames by using the **merge()** function. The data frames must have same column names on which the merging happens.
- In the example below, we consider the data sets about Diabetes in Pima Indian Women available in the library names "MASS". we merge the two data sets based on the values of blood pressure("bp") and body mass index("bmi"). On choosing these two columns for merging, the records where values of these two variables match in both data sets are combined together to form a single data frame.

Merging Data Frames

- Example

```
library(MASS)
merged.Pima <- merge(x = Pima.te, y = Pima.tr,
                     by.x = c("bp", "bmi"),
                     by.y = c("bp", "bmi")
)
print(merged.Pima)
nrow(merged.Pima)
```

	bp	bmi	npreg.x	glu.x	skin.x	ped.x	age.x	type.x	npreg.y	glu.y	skin.y	ped.y	age.y	type.y
1	60	33.8	1	117	23	0.466	27	No	2	125	20	0.088	31	No
2	64	29.7	2	75	24	0.370	33	No	2	100	23	0.368	21	No
3	64	31.2	5	189	33	0.583	29	Yes	3	158	13	0.295	24	No
4	64	33.2	4	117	27	0.230	24	No	1	96	27	0.289	21	No
5	66	38.1	3	115	39	0.150	28	No	1	114	36	0.289	21	No
6	68	38.5	2	100	25	0.324	26	No	7	129	49	0.439	43	Yes
7	70	27.4	1	116	28	0.204	21	No	0	124	20	0.254	36	Yes
8	70	33.1	4	91	32	0.446	22	No	9	123	44	0.374	40	No
9	70	35.4	9	124	33	0.282	34	No	6	134	23	0.542	29	Yes
10	72	25.6	1	157	21	0.123	24	No	4	99	17	0.294	28	No
11	72	37.7	5	95	33	0.370	27	No	6	103	32	0.324	55	No
12	74	25.9	9	134	33	0.460	81	No	8	126	38	0.162	39	No
13	74	25.9	1	95	21	0.673	36	No	8	126	38	0.162	39	No
14	78	27.6	5	88	30	0.258	37	No	6	125	31	0.565	49	Yes
15	78	27.6	10	122	31	0.512	45	No	6	125	31	0.565	49	Yes
16	78	39.4	2	112	50	0.175	24	No	4	112	40	0.236	38	No
17	88	34.5	1	117	24	0.403	40	Yes	4	127	11	0.598	28	No

Merge: Example

```
> ID<-c(1,2,3,4)
> name<-c('A','B','C','D')
> score<-c(60,70,80,90)
> student1<-data.frame(ID,name)
> student1
  ID name
1  1    A
2  2    B
3  3    C
4  4    D
> student2<-data.frame(ID,score)
> student2
  ID score
1  1    60
2  2    70
3  3    80
4  4    90
```

```
> total_student1<-merge(student1,student2,by='ID')
> total_student1
  ID name score
1  1    A    60
2  2    B    70
3  3    C    80
4  4    D    90
```

Melting and Casting

- One of the most interesting aspects of R programming is about changing the shape of the data in multiple steps to get a desired shape. The functions used to do this are called **melt()** and **cast()**.
- We consider the dataset called ships present in the library called "MASS".

Melting and Casting

```
library(MASS)
print(ships)
```

```
      type year   period   service   incidents
1     A   60     60        127         0
2     A   60     75         63         0
3     A   65     60       1095         3
4     A   65     75       1095         4
5     A   70     60       1512         6
.............
.............
8     A   75     75        2244        11
9     B   60     60       44882        39
10    B   60     75       17176        29
11    B   65     60       28609        58
.............
.............
17    C   60     60       1179         1
18    C   60     75        552         1
19    C   65     60        781         0
.............
.............
```

Melt the Data

the variable names for the columns

```
molten.ships <- melt(ships, id = c("type","year"))
print(molten.ships)
```

	type	year	period	service	incidents
1	A	60	60	127	0
2	A	60	75	63	0
3	A	65	60	1095	3
4	A	65	75	1095	4
5	A	70	60	1512	6
.....					
.....					
8	A	75	75	2244	11
9	B	60	60	44882	39
10	B	60	75	17176	29
11	B	65	60	28609	58
.....					
.....					
17	C	60	60	1179	1
18	C	60	75	552	1
19	C	65	60	781	0
.....					
.....					

	type	year	variable	value
1	A	60	period	60
2	A	60	period	75
3	A	65	period	60
4	A	65	period	75
.....				
.....				
9	B	60	period	60
10	B	60	period	75
11	B	65	period	60
12	B	65	period	75
13	B	70	period	60
.....				
.....				
41	A	60	service	127
42	A	60	service	63
43	A	65	service	1095
.....				
.....				
70	D	70	service	1208
71	D	75	service	0
72	D	75	service	2051
73	E	60	service	45
74	E	60	service	0
75	E	65	service	789
.....				

p.s. The molten variables would be the variable period, service, and incidents.

Exercise: Melt the Data

Exercise: Write a melt() api to transfer the data set on the left hand side to the right hand side

	ID	Time	X1	X2
1:	NA	1	5	NA
2:	1	2	3	5
3:	2	NA	NA	1
4:	2	1	2	4

	ID	Time	variable	value
1:	NA	1	X1	5
2:	1	2	X1	3
3:	2	NA	X1	NA
4:	2	1	X1	2
5:	NA	1	X2	NA
6:	1	2	X2	5
7:	2	NA	X2	1
8:	2	1	X2	4

Cast the Molten Data

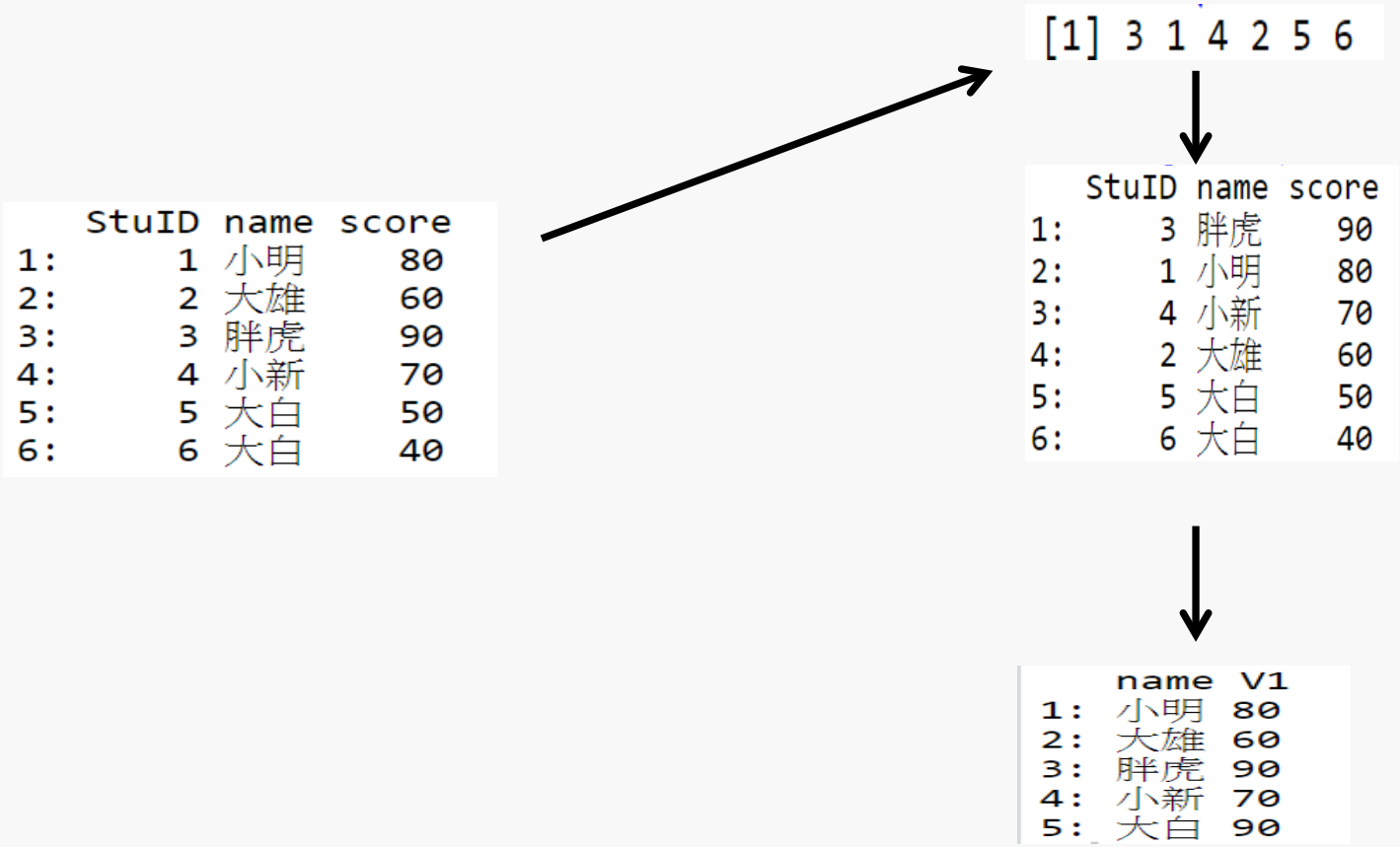
- We can cast the molten data into a new form where the aggregate of each type of ship for each year is created. It is done using the **cast()** function.

```
recasted.ship <- dcast(molten.ships, type+year~variable,sum)
print(recasted.ship)
```

```
      type year  variable  value
1        A   60    period     60
2        A   60    period     75
3        A   65    period     60
4        A   65    period     75
............
............
9        B   60    period     60
10       B   60    period     75
11       B   65    period     60
12       B   65    period     75
13       B   70    period     60
............
............
41       A   60   service    127
42       A   60   service     63
43       A   65   service   1095
............
............
```

	type	year	period	service	incidents
1	A	60	135	190	0
2	A	65	135	2190	7
3	A	70	135	4865	24
4	A	75	135	2244	11
5	B	60	135	62058	68
6	B	65	135	48979	111
7	B	70	135	20163	56
8	B	75	135	7117	18
9	C	60	135	1731	2
10	C	65	135	1457	1
11	C	70	135	2731	8
12	C	75	135	274	1
13	D	60	135	356	0
14	D	65	135	480	0
15	D	70	135	1557	13
16	D	75	135	2051	4
17	E	60	135	45	0
18	E	65	135	1226	14
19	E	70	135	3318	17
20	E	75	135	542	1

Exercise: order()



Exercise

- Sort the values of the “Food” column on LHS to gain the new table on the RHS. Attention pays to that the corresponding values of other columns also needs to move together on the same row.

	First	Second	Food
1	0	-2	apple
2	1	-1	orange
3	2	0	hottea
4	3	1	berry
5	4	2	melon
6	5	3	grape

	First	Second	Food
1	0	-2	apple
4	3	1	berry
6	5	3	grape
3	2	0	hottea
5	4	2	melon
2	1	-1	orange

The End