

Module 3

Data Manipulation in R

Module 3: Data Manipulation in R

No.	Topics	Estimated Time (min)
1	Control Flow	60
2	Tables and cross-tabulation	60
3	Data Input and Data Management Input/Output Reshaping Data Date & Time	60
4	Dealing with Missing Data	30
5	String Manipulation	60

1. Control Flow

- Control flow

- Looping:

- 3 types of loops: for loop, while loop, repeat loop

- Command:

- next: to skip the loop but continue looping
 - break: quit

- for loop:

```
for(i in 1:5)
{ print(i)
  print("Hello world!")
  cat(sprintf("Hello World %s\n", i))
}
```

- For each i, taking each value start from 1 till 5. We can also declare a vector, and let i take each of the components of the vector, where first iteration is equal to the first component of the vector; second iteration is equal to the second component and so on

Control Flow cont...

- While and repeat are c-style looping

- while loop

```
i<- 1
```

```
while(i<=10)
```

```
{ i<- i + 4}
```

```
i #the last i value is 13,
```

– you can add in print to see how i is changed

- repeat (must use with break to stop the loop)

```
i<- 1
```

```
repeat
```

```
{ i<- i + 4
```

```
  print(i)
```

```
  if(i > 10) break
```

```
}
```

Control Flow cont...

– Looping over non-vector set

- R not really support iteration over non-vector set, but can accomplish from: `lapply()` and `get()`
- `lapply()` assume that iterations of the loop are independent of each other
- `get()` takes as an argument a character string representing the name of some object and returns the object of that name
- Create two matrices (non-vector) as follow:

```
u<-matrix(c(1,2,3,1,2,4), nrow=3, ncol=2)  
v<-matrix(c(8,12,20,15,10,2), nrow=3, ncol=2)
```
- We want to apply linear regression function to each of them

```
for(m in c("u","v")) {  
  z<-get(m) print(lm(z[,2] ~z[,1]))  
}  
# m was first set to u, then lines assign matrix u to z  
# m was then set to v, and lines assign matrix v to z
```

Control Flow cont...

– If/Else

- Type the following commands:
 - `x <- -5`
 - `if(x > 0) #just copy and paste the command in console`
`{ print("Non-negative number")`
`} else { print("Negative number") }`
- A compact type of if/else statement:
 - `x <- 2`
 - `y <- if(x==2) x else x+1`

Exercise 3.1 – 25 minutes

- Create a for loop so that it prints the following output:

```
1 2 3 4 5
```

```
for ( i in 1:5){  
    cat(i, " ")  
}
```
- Write a function which scan in the marks and return an appropriate grade by using if/else
 - 80 and above, grade = A
 - 60-79, grade = B
 - 50-69, grade = C
 - Below 50, grade = fail

Tables and cross-tabulation

- `apply()` family
 - Form the basis of more complex combinations and helps to perform operations with very few lines of code
 - Returns a vector or array or list of values obtained by applying a function to margins of an array or matrix
 - `apply()`: can be used for data frame or matrix; return a vector, list or array
 - `lapply()` – to apply a given function (FUN) to every element of a list and obtain a list as result
 - Can be used for dataframes, list, or vectors
 - Return a list with same number of elements as the object passed to it
 - Try the following commands:
 - `A <- matrix(c(1,2,3,4,5,6,7,8,9),nrow=3,ncol=3)`
 - `B <- matrix(c(4,5,6,7,8,9,10,11,12,13,14,15),nrow=4,ncol=3)`
 - `C <- matrix(c(8,9,10,8,9,10),nrow=3,ncol=2)`
 - `MyList<-list(A,B,C)`
 - `lapply(MyList,"[, 2)`
 - » Extract the 2nd column from `MyList`, `[` is a standard selection operator in R
 - `lapply(MyList,"[, 1, 2) # to extract a single element from each matrix`
 - `lapply(MyList,"[, 1,) # Extract the 1st row from `MyList``

Tables and cross-tabulation

- Lapply

- Lets look at the power of lapply
 - Open the function Normal.R
 - Observe the code
 - Compare the result you obtain from Normal.R with the following command:
 - `df1<-as.data.frame(lapply(d,function(x){(x-min(x))/(max(x)-min(x))}))`

Tables and cross-tabulation

- `apply()` family (cont...)
 - `apply()` - function works like `lapply()`, but it tries to simplify the output to the most elementary data structure that is possible. And indeed, `apply()` is a 'wrapper' function for `lapply()`.
 - Can be used for dataframe, list, or vector
 - Return a vector or matrix

```
# Return a vector with `sapply()`  
sapply(MyList,"[, 2, 1 )  
  
# Return a list with `sapply()`  
sapply(MyList,"[, 2, 1, simplify=F)
```

Tables and cross-tabulation cont...

- `apply()` family (cont...)
 - `mapply()` - 'multivariate' `apply`. Its purpose is to be able to vectorize arguments to a function that is not usually accepting vectors as arguments.

```
# Create a 4x4 matrix
```

```
Q1 <- matrix(c(rep(1, 4), rep(2, 4), rep(3, 4), rep(4, 4)), 4, 4)
```

```
# Print `Q1`
```

```
print(Q1)
```

```
# Or use `mapply()`
```

```
Q2 <- mapply(rep, 1:4, 4)
```

```
print(Q2) # Print `Q2`
```

Tables and cross-tabulation cont...

- `tapply()` – computes a measure (mean, median, min, max, etc..) or a function for each factor variable in a vector.
 - `#generate some fake data set`
 - `medical.example <- data.frame(patient = 1:100, age = rnorm(100, mean = 60, sd = 12), treatment = gl(2, 50, labels = c("Treatment", "Control")))`
 - `medical.example` # print out the data set
 - `summary(medical.example)` #summarise the data
 - `tapply(medical.example$age, medical.example$treatment, mean)`
 - # the result provide average age of patients
 - # who have “control” and those who have “treatment”

Tables and cross-tabulation cont...

- Create a table
 - `smoke <- matrix(c(51,43,22,92,28,21,68,22,9),ncol=3,byrow=TRUE)`
 - `colnames(smoke) <- c("High","Low","Middle")`
 - `rownames(smoke) <- c("current","former","never")`
 - `smoke <- as.table(smoke)`
 - `smoke` #or `fTable(smoke)` to print out the values of table
- Count the marginal distribution
 - `margin.table(smoke)`
 - `margin.table(smoke,1)` #row alternative command: `apply(smoke,1,sum)`
 - `margin.table(smoke,2)` #column alternative command: `apply(smoke,2,sum)`
- Get the proportion:
 - `smoke/margin.table(smoke)`
 - `margin.table(smoke,1)/margin.table(smoke)`
 - `margin.table(smoke,2)/margin.table(smoke)`
 - `prop.table(smoke)` #cell percentage
 - `prop.table(smoke,1)` #row percentage
 - `prop.table(smoke,2)` #column percentage
- Get a summary report of table `smoke`:
 - `summary(smoke)`

	High	Low	Middle
current	51	43	22
former	92	28	21
never	68	22	9

Tables and cross-tabulation cont...

- Compute the contingency table:
 - `x<- factor(c(" yes ", " yes ", " no ", " not sure ", " no "))`
 - `y<- factor(c(" yes ", " no ", " no ", " yes ", " no "))`
 - `d<-data.frame(x,y)`
 - `cttab<-table(d)`
 - `cttab[1,]`
 - `cttab/5` #obtain the proportion
 - `apply(cctab,1,sum)`
 - `addmargins(cttab)` #find marginal total

Tables and cross-tabulation cont...

- Use the medical.example

- Condition check

- ```
#to get no of patients with age<80
```

- ```
table(medical.example$age<80)
```

- ```
#to obtain no of labels of each treatment category
```

- ```
table(medical.example$treatment)
```

- ```
#check whether any missing values
```

- ```
is.na(medical.example$treatment)
```

- Sorting, Ranking and ordering

- `order()` is better than `sort()` as `sort()` will uncouple the values that suppose in the same record.
 - **Order** returns the position of the original value and is in the order of **sorted** sequence, that is smallest value to largest value
 - **Rank** references the position of the value in the sorted vector and is in the same order as the **original** sequence . In other words, it returns the order of each element in an ascending list
 - `ordered<-order(medical.example$age)` #order the age in ascending order
 - `#order` will return the index order value
 - `#display` based on ordered row index
 - `medical.example[ordered,]`
 - `ordered<-order(-medical.example$age)` # order the age in descending order
 - `ranked<-rank(medical.example$age)`
 - `cbind(medical.example,ranked)` #let's see how the record is ranked
 - `sorted<-sort(medical.example$age)`
 - `x<-1:5` #create a x variable
 - `rev(x)` # reverse the order of x element
 - `rev(sort(medical.example$age))` # to sort in descending order
 - `sort(medical.example$age, decreasing = TRUE)`

Exercise 3.2 – 20 minutes

- Given the following R command:
 - `baseball.example <- data.frame(team = gl(5, 5, labels = paste("Team", LETTERS[1:5])), player = sample(letters, 25), batting.average = runif(25, .200, .400))`
- Based on the data above, please solve the following questions:
 - Provide a summary report of the data
 - Provide the maximum batting of each team and list out the result
 - Order the record based on team and batting.average
 1. `summary(baseball.example)`
 2. `tapply(baseball.example$batting.average, baseball.example$team, max)`
 3. `baseball.example[order(baseball.example$team, baseball.example$batting.average) ,]`

3. Data Input and Data Management: Input/Output

- Preparing data for analysis may consume more time than analyzing the data
- Various sources can be imported to R

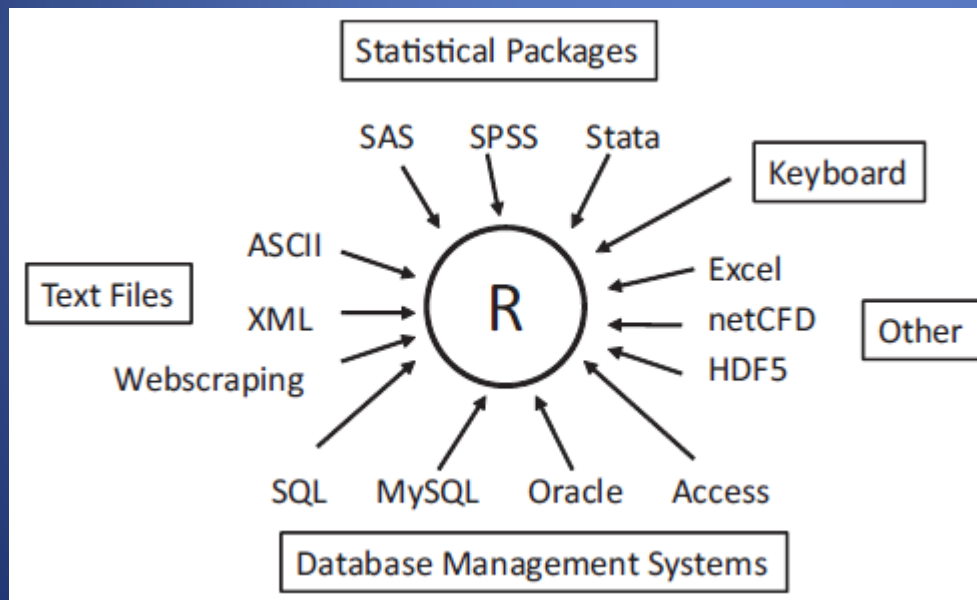


Figure: Sources that can be imported to R (Kabacoff, 2011)

Data Input and Data Management: Input/Output cont...

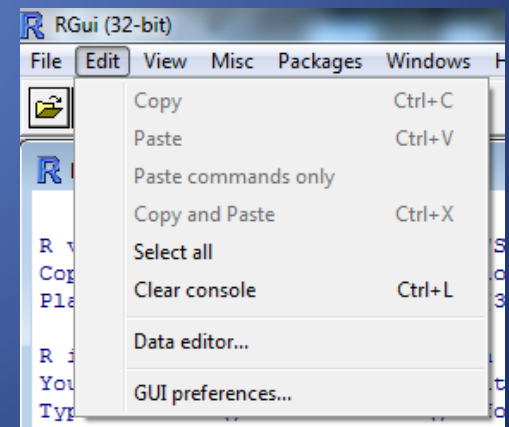
- Entering Data

- Data Editor of R (to edit existing data frame)

- Use RGui

```
medical.example <- data.frame(patient = 1:100, age =  
rnorm(100, mean = 60, sd = 12), treatment = gl(2, 50, labels  
= c("Treatment", "Control")))
```

- Click->Edit->Data Editor
 - You can edit any values in the editor



Data Input and Data Management: Input/Output cont...

- Read in tabular format:

Function	Description
<code>read.table()</code>	main function to read files in table format
<code>read.csv()</code>	reads csv files separated by a comma ","
<code>read.csv2()</code>	reads csv files separated by a semicolon ";"
<code>read.delim()</code>	reads files separated by tabs "nt"
<code>read.delim2()</code>	similar to <code>read.delim()</code>
<code>read.fwf()</code>	read fixed width format files

- When reading data with small to moderately sized datasets, `read.table` can be called without specifying any other parameters. For e.g.: `read.table("datasets.txt")`

Data Input and Data Management: Input/Output cont...

- To read in huge datasets:
 - Read the help page of `read.table` so that you know how to set the parameters
 - Roughly calculate the memory to store the dataset. If the needed size exceeds the RAM of your computer, just stop right here.
 - Set `comment.char=""` if there are no commented lines in your file.
 - Specify `colClasses` instead of using the default value will make 'read.table' run faster. If all columns are numeric format, then `colClasses="numeric"`
 - Specify `colTypes` can avoid if readr guessing wrongly on the column types

colTypes	Descriptions
<code>col_integer()</code>	to specify integer (alias = "i")
<code>col_double()</code>	to specify double (alias = "d").
<code>col_logical()</code>	to specify logical variable (alias = "l")
<code>col_character()</code>	leaves strings as is. Don't convert it to a factor (alias = "c").
<code>col_factor()</code>	to specify a factor (or grouping) variable (alias = "f")
<code>col_skip()</code>	to ignore a column (alias = "-" or "_")
<code>col_date()</code>	(alias = "D"), <code>col_datetime()</code> (alias = "T") and <code>col_time()</code> ("t") to specify dates, date times, and times.

Data Input and Data Management: Input/Output cont...

- Example of command to set colTypes
 - `read_csv("my_file.csv", col_types = cols(x = "i", # integer column
treatment = "c" # character column))`
- Example of memory calculation: suppose you have 100 row with 20 columns, all are numerics
 - $100 * 20 * 8 \text{ bytes (for numeric)} = 16000 \text{ bytes} = 16 \text{ kb}$

Data Input and Data Management: Input/Output cont...

- Reading data .csv file
 - Each line represent one case
 - All lines have the same number of values, separated by commas
 - First value in each line may be the case name
 - Software that deals with rectangular data set (rows and columns) such as Excel, SPSS are capable of reading and writing CSV file
 - Simplest method to move data from one program to another
 - Character data field:
 - Need not to enclose with quotes (e.g. in strongly agree instead of “in strongly agree”)
 - Unless the field has commas then must be quoted (e.g.: “don’t know, refused” instead of don’t know, refused)
 - Either single quote or double quotes can be used but not the mix of both (e.g. “don’t know, refused’ is illegal)
 - Try the following steps:
 - `getwd()` # to identify current working directory of R
 - #copy paste this file: Datasets.xlsx to the working directory
 - `library(readxl)` #need to install readxl
 - `myData<-read_excel("Datasets.xlsx")`
 - `myData` #print the data

Data Input and Data Management: Input/Output cont...

- Reading data .txt or .csv file
 - # Read tab separated values `read.delim(file.choose())`
 - E.g.:
 - Copy Duncan.txt to the working directory of R
 - `Data1<-read.delim("Duncan.txt")`
 - Data1 #print out the data
 - # Read comma (",") separated values `read.csv(file.choose())`
 - # Read semicolon (";") separated values `read.csv2(file.choose())`
 - #to read a text file one line at a time or in a single operation
`x<-readLines("Module3_scan.txt")` #warning is generated
x
 - The readLines command return the result as a vector of strings
 - Can check the dimension of a data set:
 - `dim(Data1)` #it will return no of rows and columns

Data Input and Data Management: Input/Output cont...

- Reading data with scan
 - Appropriate to use when all data to be read is of the same mode
 - Can read numeric or characters from a file or from keyboard
 - By default, scan assume all the read data is numeric
 - If no argument is given, it will read from console
 - Type the following command:

```
names = scan(what="")
```

```
1: joe fred bob john #key in the name and press <enter>
```

```
5: sam sue robin #key in the name and press<enter>
```

```
8: # press <enter>
```

```
names #print the values
```

```
names = scan(what=list(a=0,b="",c=0)) #scan in 3 columns which is a list
```

```
1: 1 dog 3 #key in 1 dog 3 <enter>
```

```
2: 2 cat 5
```

```
3: 3 duck 7 4:
```

```
names
```

Data Input and Data Management: Input/Output cont...

- Try to scan a text file:

```
scan("Module3_scan1.txt")
```

```
scan("Module3_scan.txt") #this will generate error
```

- The error is due to the nonnumeric contents of the text file, an extra command `what=""` need to include. Character string will assign to `what`.

```
scan("Module3_scan.txt",what="")
```

- Embedded a scan in matrix

```
mymat = matrix(scan(),ncol=3,byrow=TRUE)
```

```
1: 19 17 12 <enter>
```

```
4: 15 18 9 <enter>
```

```
7: 9 10 14 <enter>
```

```
10: 7 12 15 <enter>
```

```
13: <enter>
```


Data Input and Data Management: Input/Output cont...

- Reading files with mixture of numeric, character and header

```
x<-read.table("Duncan.txt",header=TRUE)
#read.table will return a data frame
#it expects each line with similar number of fields
#header = TRUE means that the input data consists of variable names separated by the same separator
```
- Reading through file

```
x<-file("Module3_scan.txt","r")
readLines(x,n=1) #print the result of first line
#use of while loop to read all lines
x<-file("Module3_scan.txt","r")
while(TRUE){
  r1<-readLines(x,n=1)
  if(length(r1)==0){
    print("reached the end") #through detect EOF in file
    break
  } else print(r1)
}
```
- We can reset the position of file to the start of the file through seek()

```
readLines(x,n=1) #after the while loop, check the file position
seek(con=x,where=0)
#where=0 means that we want to position file pointer start at the beginning
readLines(x,n=1)
```


Data Input and Data Management: Input/Output cont...

- Connections: flexible way for R to read data from various sources

Function	Data source
file	Files on the local file system
pipe	Output from a command
textConnection	Treats text as a file
gzfile	Local gzipped file
unz	Local zip archive (with single file;read-only)
bzfile	Local bzipped file
url	Remote file read via http
socketConnection	socket for client/server programs

Data Input and Data Management: Input/Output cont...

- When a connection object is created, it does not automatically open the object
- Access file via URLs
 - Some I/O function, such as `read.table()` and `scan()` accept web URLs as arguments
 - `x<-"https://archive.ics.uci.edu/ml/machine-learning-databases/"`
 - `x<-paste(x,"iris/iris.data",sep="")`
 - `Data1<-read.csv(x)`
 - `head(Data1)` #display the first 6 records
 - `tail((Data1)` #display the last 6 records
 - `tail(Data1, n =10)` #display the last 10 records
 - Another alternative:
 - `x<-read.csv(stringsAsFactors = FALSE, url("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"))`

Data Input and Data Management: Input/Output cont...

- File function:
 - `con<-file("Iris.txt")` #create a connection to Iris.txt
 - `open(con, "r")` #open connection in read mode
 - `data <- read.delim(con)` #read from connection
 - `close(con)` #close the connection
- Open parameters for file function:
 - “r” open file in read only mode
 - “w” open a file for writing (and initializing a new file)
 - “a” open a file for appending
 - “rb”, “wb”, “ab” reading, writing, or appending in binary mode (Windows)
- Calculate the number of lines in the file using this command:
 - For Windows user: useful when you want to specify number of rows in read command
`nrec = as.numeric(shell('type "Iris.txt" | find /c ",", intern=TRUE))`
`nrec`
 - For UNIX user
`nrec = as.numeric(system('cat comma.txt | wc -l', intern=TRUE))`
- Close a connection after use:

Data Input and Data Management: Input/Output cont...

- Using `dput()` and `dget()`
 - Deparsing R object with `dput()` and reading it back with `dget()`
 - `dput(medical.example)` #output is in the form of R code
 - #send output of `dput` to a file
 - `dput(medical.example, file="medical.R")`
 - #read in the output
 - `new.medical<-dget("medical.R")`
- With Azure ML:
 - We can output R data frame as a rectangle table with the following command:
 - `maml.mapOutputPort('iris')` #not for RStudio

Data Input and Data Management: Input/Output cont...

- GUI: Importing data from plain text, i.e. .txt file
 - Click File->Import Data Set->Text file (base)->Duncan.txt
 - Click Import

Import Dataset

Name: Duncan

Encoding: Automatic

Heading: ☒ Yes ☐ No

Row names: Automatic

Separator: Whitespace

Decimal: Comma

Quote: Double quote (")

Comment: None

na.strings: NA

☒ Strings as factors

Input File

	type	income	education	prestige
accountant	prof,tech,manag	62	86	86
pilot	prof,tech,manag	72	76	76
architect	prof,tech,manag	75	92	92
author	prof,tech,manag	55	90	90
chemist	prof,tech,manag	64	86	86
minister	prof,tech,manag	21	84	84
professor	prof,tech,manag	64	93	93
dentist	prof,tech,manag	80	100	100
reporter	white-collar	67	87	87
engineer	prof,tech,manag	72	86	86
undertaker	prof,tech,manag	42	74	74
lawyer	prof,tech,manag	76	98	98
physician	prof,tech,manag	76	97	97
welfare.worker	prof,tech,manag	41	84	84
teacher	prof,tech,manag	48	91	91
conductor	white-collar	76	34	34

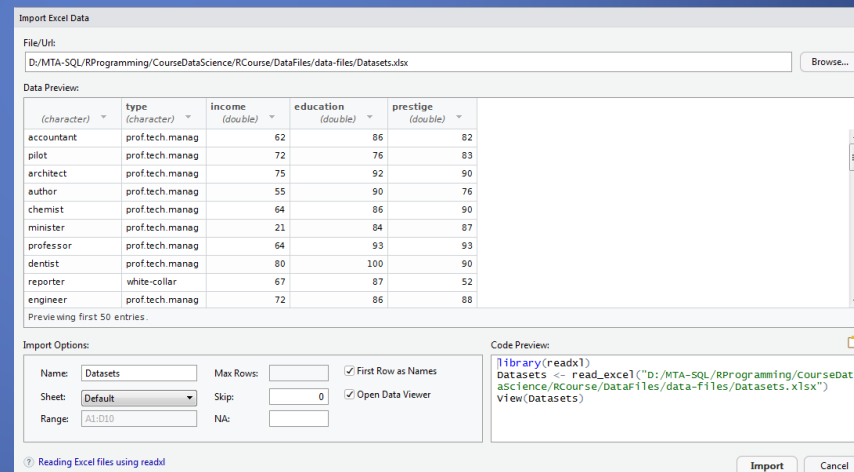
Data Frame

type	income	education	prestige
prof,tech,manag	62	86	82
prof,tech,manag	72	76	83
prof,tech,manag	75	92	90
prof,tech,manag	55	90	76
prof,tech,manag	64	86	90
prof,tech,manag	21	84	87
prof,tech,manag	64	93	93
prof,tech,manag	80	100	90
white-collar	67	87	52
prof,tech,manag	72	86	88
prof,tech,manag	42	74	57
prof,tech,manag	76	98	89
prof,tech,manag	76	97	97
prof,tech,manag	41	84	59
prof,tech,manag	48	91	73
white-collar	76	34	38
prof,tech,manag	53	45	76
prof,tech,manag	60	56	81

Import Cancel

Data Input and Data Management: Input/Output cont...

- GUI: Importing data from spreadsheets
 - Need to install readxl package
 - After done, you will see the following figure. Just browse to import the excel data set:



- Can import data files from SPSS, SAS and sdata

Data Input and Data Management: Input/Output cont...

- Writing Data
 - #load a R data set
 - `data(mtcars)`
 - `write.table(mtcars, file = "mtcars.txt", sep = "\t", row.names = TRUE, col.names = NA)`
 - #the file may save in "C:\Users\Documents"
- **`write.csv()`** uses "." for the decimal point and a comma (",") for the separator.
- **`write.csv2()`** uses a comma (",") for the decimal point and a semicolon (";") for the separator.

Data Input and Data Management: Input/Output cont...

- Writing data into different format
 - `install.packages("readr")`

```
opened URL
downloaded 1004 Kb

package 'curl' successfully unpacked and MD5 sums checked
warning in install.packages :
  unable to move temporary installation 'C:\Users\PYGoh\Documents\R\win-library\3.1\file51423055cc\curl' to 'C:\Users\PYGoh\Documents\R\win-library\3.1\curl'
package 'BH' successfully unpacked and MD5 sums checked
package 'readr' successfully unpacked and MD5 sums checked
```

- `library("readr")`
- `# Writing mtcars data to a tsv file`
- `write_tsv(mtcars, path = "mtcars.tsv")`
- `# Writing mtcars data to a csv file`
- `write_csv(mtcars, path = "mtcars.csv")`

Data Input and Data Management: Input/Output cont...

- Saving and loading data
 - #save the whole workspace
 - `save.image(file="Workspace1.RData")`
 - # Save a single object to a file
 - `saveRDS(mtcars, "mtcars.rds")`
 - # Restore it under a different name
 - `my_data <- readRDS("mtcars.rds")`
 - # Saving on object in RData format
 - `save(object1, file = "data1.RData")`
 - # Save multiple objects
 - `save(object1, object2, file = "data2.RData")`
 - # To load the data in the saved workspace
 - `load("data1.RData")`

Exercise 3.3a – 15 minutes

- Assign the following URL to x:
https://jozefhajnal.gitlab.io/r/post/data/ESA2010_GDI.csv
- Try to read the data and assign it into Data2.
- Save this Data2 as a file with RData format.

```
Data2 <- read.csv("ESA2010_GDI.csv")
```

```
save(Data2, file = "data2.RData")
```

```
y<- scan("Module3_scan.txt", what="")
```

Exercise 3.3b – 15 minutes

- Store the scanned result of Module3_scan.txt in a variable called y
- Print out the 3rd element of y

Data Input and Data Management:

Reshaping Data cont...

- Data may need to modify to suit the R functions.
- Since most R function deal with data frames, so we will focus on the working with data frames.
- Use a dataset called Loblolly which records the height, age and seed information of pine trees.
- To remain the data in system version but change the data in workspace:
 - `Loblolly$logheight = log(Loblolly$height)`
 - Create a new column which is called logheight
 - `Loblolly = transform(Loblolly, logheight = log(height))`
 - Now the local workspace Loblolly will have a new column, and this new column values are having similar height values of the original data

Data Input and Data Management: Reshaping Data

- Transform data into TRUE/FALSE
 - `bigsepal = iris$Sepal.Length > 6`
 - `Sepal.Length > 6` is true, less than is false
- More categories:
 - `sepalgroup = 1 + (iris$Sepal.Length >= 5) + (iris$Sepal.Length >= 7)`
 - Category 1, if ≤ 5 ; label as 2 if $5 < x < 7$; label as 3 if ≥ 7
 - `sepalgroup3 = ifelse(iris$Sepal.Length < 5, 1, ifelse(iris$Sepal.Length <= 7, 2, 3))`
 - Or using if/else to transform the data

Data Input and Data Management: Reshaping Data

- Converting objects:

Conversion to	Function	Rules
numeric	as.numeric	FALSE -> 0
		TRUE -> 1
		"1", "2", ... -> 1, 2, ...
		"A", ... -> NA
logical	as.logical	0 -> FALSE
		other numbers -> TRUE
		"FALSE", "F" -> FALSE
		"TRUE", "T" -> TRUE
		other characters -> NA
character	as.character	1, 2, ... -> "1", "2", ...
		FALSE -> "FALSE"
		TRUE -> "TRUE"

Data Input and Data Management: Reshaping Data

- Converting objects (cont...):
 - `gender<- factor(c("Male", "Female"))`
 - `gender`
 - Output: [1] Male Female Levels: Female Male
 - `as.numeric(gender)`
 - [1] 2 1

Data Input and Data Management:

Reshaping Data

- Commonly, data is in wide format. However, some analysis and visualization tool needs the data in long format. Thus, reshaping is needed.
- Package: reshape2 – to transform data between wide and long formats
 - Easier to produce a pivot table

- Wide format:

```
##      ozone      wind      temp
## 1 23.61538 11.622581 65.54839
## 2 29.44444 10.266667 79.10000
## 3 59.11538  8.941935 83.90323
## 4 59.96154  8.793548 83.96774
```

- Long format:

```
## No id variables; using all as measure variables

##   variable      value
## 1    ozone 23.615385
## 2    ozone 29.444444
## 3    ozone 59.115385
## 4    ozone 59.961538
## 5     wind 11.622581
## 6     wind 10.266667
## 7     wind  8.941935
## 8     wind  8.793548
## 9     temp 65.548387
## 10    temp 79.100000
## 11    temp 83.903226
## 12    temp 83.967742
```


Data Input and Data Management: Reshaping Data

- Reshape2 (cont...):
 - 2 main functions:
 - melt takes wide format data and melt it into long format data
 - cast takes long format data and cast it into wide format data
 - Try the following command:
 - library(reshape2)
 - names(airquality) <- tolower(names(airquality))
 - head(airquality) #original data is like this
 - aql <- melt(airquality)
 - head(aql)
 - aql <- melt(airquality, id.vars = c("month", "day"), variable.name = "climate_variable", value.name = "climate_value")
 - To name the columns with user define column's name, we can do it through variable.name and value.name
 - We can let melt know that what columns we want it to be the ID variables. This ID variables are the variables that identify individual rows of data (like a primary key in database)
 - head(aql)

Data Input and Data Management: Reshaping Data

- Reshape2 (cont...):
 - There are various cast function. For e.g.: dcast is to deal with data frame, acast is to deal with vector, matrix or array.
 - Type the following command:
 - `aql <- melt(airquality, id.vars = c("month", "day")) #cont`
 - `aqw <- dcast(aql, month + day ~ variable)`
 - `head(aqw)`
 - `dcast(aql, month ~ variable, fun.aggregate = mean, na.rm = TRUE)`
 - This one work like the Group By in database
 - Function mean is applied, where the mean value is by month. For e.g.: month 5, then it produces the result for each climate
 - na.rm is a function to exclude missing values in calculation

Data Input and Data Management:

Reshaping Data cont...

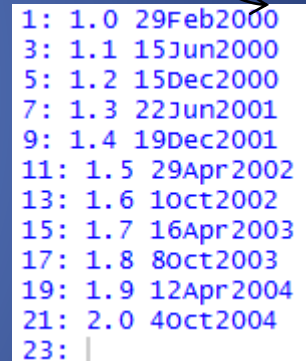
- R provides basic reshape without the need of third party package too
- This reshape function take in 3 important arguments: data(i.e. the data frame), direction(either wide or long), varying (the repeated measures columns that we want to stack/arrange)
 - `Data<-read.delim("DataReshape.txt",sep="")`
 - `reshape(Data, direction="long", varying=7:9)`
 - This gives error due to columns naming, the system not able to read it properly
 - `names(Data)[7:9] <- c("time.1", "time.2", "time.3")`
 - Rename the data
 - `reshape(Data, direction="long", varying=7:9)`
 - Time should be the measurement and now the time data is messy
 - Human could not understand well
 - `reshape(Data, direction="long", varying=7:9, idvar='id', timevar="TIME",v.names="RESULTS")`
 - Put in new parameter to make the data more meaningful
 - timevar: this shows the repeated times of measure (time.1, time.2, time.3)
 - v.names: the column's name of the repeated measures
 - id.var: the row id

Data Input and Data Management: Date & Time

- Conversion of date and time:
 - Assume if your date information is stored as second:
 - `dt = c(1127056501, 1104295502, 1129233601, 1113547501, 1119826801, 1132519502, 1125298801, 1113289201)`
 - `mydates = dt`
 - `class(mydates) = c('POSIXt','POSIXct')`
 - `mydates`
 - `mydates = structure(dt,class=c('POSIXt','POSIXct'))`
 - `mydate = strptime('16/Oct/2005:07:51:00',format='%d/%b/%Y:%H:%M:%S')`
 - To extract specific day information:
 - `bdays = c(tukey=as.Date('1915-06-16'),fisher=as.Date('1890-02-17'),cramer=as.Date('1893-09-25'), kendall=as.Date('1907-09-06'))`
 - `weekdays(bdays)`

Data Input and Data Management: Date & Time

- Operations with date and time:
 - `rdates = scan(what="")`
 - `scan` in a series of release date time information of R (see Figure)
 - `rdates = as.data.frame(matrix(rdates,ncol=2,byrow=TRUE))`
 - Make it a data frame
 - `rdates[,2] = as.Date(rdates[,2],format='%d%b%Y')`
 - `names(rdates) = c("Release","Date")`
 - Provide columns' name
 - `mean(rdates$Date)`
 - `range(rdates$Date)`
 - `difftime(rdates$Date[11],rdates$Date[1],units='weeks')`
 - `table(format(rdates$Date,'%A'))`
 - Look at the distribution in weekdays



```
1: 1.0 29Feb2000
3: 1.1 15Jun2000
5: 1.2 15Dec2000
7: 1.3 22Jun2001
9: 1.4 19Dec2001
11: 1.5 29Apr2002
13: 1.6 10Oct2002
15: 1.7 16Apr2003
17: 1.8 8Oct2003
19: 1.9 12Apr2004
21: 2.0 4Oct2004
23: |
```


Dealing with Missing Data

- Two kinds of missing data:
 - Missing completely at random
 - Small amount, acceptable, just remove from analysis
 - Huge amount, could be a problem
 - Threshold: maximum 5% for large data set
 - If more than 5%, drop the feature or remove the sample
 - Missing not at random (serious issue)
 - If data is collected from survey, then need to go back to survey and investigate the questions again
- Deal with missing data: convert '?' to NA; remove NA rows
 - Most of the statistical summary has na.rm function
 - If na.rm is set as TRUE, it will remove the missing value before calculate the statistical function
 - '!' operator can be used to ignore missing values during the creating of vector:
 - E.g.: `(!is.na(x))`
 - Importing data from external sources:
 - Missing value may be labelled as NA or some other string

Dealing with Missing Data cont...

- Commands that deal with missing data:
 - Test for missing data:
 - `is.na(x)` # returns TRUE if x is missing
 - If x is a vector where `x<-c(1,2,3,NA)`, then may returns F F F T
 - To exclude missing data from analysis
 - `mean(x, na.rm=TRUE)` # `x<-c(1,2,NA,3)`
 - Returns a logical vector indicating which cases are complete
 - `mydata[!complete.cases(mydata),]`
 - List rows of data that have missing values
 - Create a new data set from an existing data set by excluding all missing data
 - `newdata <- na.omit(mydata)`

Dealing with Missing Data cont...

- Try the following commands:
 - `Name <- c("John", "Tim", NA)`
 - `Sex <- c("men", NA, "women")`
 - `Age <- c(45, 53, NA)`
 - `dt <- data.frame(Name, Sex, Age)`
 - `dt`
 - `na.omit(dt)`
 - `data <- airquality`
 - `summary(data)` #check the summary
 - `pMiss <- function(x){sum(is.na(x))/length(x)*100}`
 - Check the proportion of missing data: any feature/sample with missing data >5%?
 - `apply(data,1,pMiss)` #check the rows
 - `apply(data,2,pMiss)` #check the columns
 - `data[,complete.cases(t(data))]` #to remove columns with missing data

Exercise 3.4a – 10 minutes

- Load Data2 (the one you save as Rdata in Exercise 3.3a)
ESA2010_GDI
- Display the first 6 rows of Data2
- Observe the varying columns index so that you know what index to set for varying
- Reshape the data with this arguments: country is the idvar, GDI is the column's name of repeated measure, timevar is year and times range from 1995 to 2016.

Exercise 3.4b – 10 minutes

- Obtain the percentage of missing values from each row of Data3
- Produce a summary report of Data3
- Remove all the missing values from Data3 and assign it to newData3. Then, produce a summary report of newData3
- Compare the number of dimensions between Data3 and newData3. Do you see the difference?

4. String Manipulation cont...

- When we mix numeric and characters in vectors or matrices, R treat it as characters:
 `A<-c(1:5, TRUE, pi, "text", FALSE)`
 `class(A)`
 - For data frame, characters are treated as factor
 - `A = data.frame(numbers = 1:5, letters = letters[1:5])`
 - `str(A)`
 - #can turn off the stringAsFactor for data frame
 - `df2 = data.frame(numbers = 1:5, letters = letters[1:5], stringsAsFactors = FALSE)`
 - `str(df2)`
- Use `medical.example` (a data frame which has created)
- To examine the structure of data frame:
 - `str(medical.example)`
- Convert to character string
 - `as.character()`
 - Example:
 - `A = 6 + 9`
 - `as.character(A)`

String Manipulation cont...

- Re-visit paste()
 - paste() – concatenate vectors after converting to character
 - paste(..., sep = " ", collapse = NULL); sep means separator, collapse is to identify whether we want to collapse all strings into a single string
 - lloveR = paste("Hello", "How are you", "R", sep = "-")
 - paste("X", 1:5, sep = ".")
 - #paste with collapse
 - paste(1:3, c("!", "?", "+"), sep = "", collapse = "")
 - #paste without collapse
 - paste(1:3, c("!", "?", "+"), sep = "")
 - #with missing value NA,
 - paste("the value of e is", exp(1), NA) # it treat NA as part of string
 - #paste0 is similar to the collapse
 - paste0("let's", "collapse", "all", "these", "words")
 - nth <- paste0(1:12, c("st", "nd", "rd", rep("th", 9)))
 - paste(month.abb, nth, sep = ": ", collapse = "; ")

String Manipulation cont...

- Re-visit print()
 - print() is useful to create and build string
 - Printing function

Function	Description
print()	generic printing
noquote()	print with no quotes
cat()	concatenation
format()	special formats
toString()	convert to string
sprintf()	printing

- print() displays text in quoted form. To off that, just set the argument quote = FALSE
 - my_string = "programming with data is fun"
 - print(my_string, quote = FALSE)
 - noquote(my_string)
 - no_quotes = noquote(c("some", "quoted", "text", "!%^(&=)")
#subscripting)

String Manipulation cont...

- `cat()` – to concatenate objects and print them either on screen or to a file
- run the following commands to see the effect
 - `cat(my_string)`
 - `cat(my_string, "with R")`
 - `cat(my_string, "with R", sep = " = ")`
 - `cat(1:10, sep = "-")`
 - `cat(month.name[1:4], sep = " ")`
 - #to print the output in a file
 - `cat(my_string, "with R", file = "output.txt")`
 - #fill is a parameter to specify the max no of characters allow in one line
 - `cat("Looooooooooooong strings", "can be displayed", "in a nice format", "by using the fill argument", fill = 30)`

String Manipulation cont...

- `format()` - format an R object for pretty printing.
- Useful parameters:
 - width the (minimum) width of strings produced
 - trim if set to TRUE there is no padding with spaces
 - justify controls how padding takes place for strings. Takes the values "left", "right", "centre", "none". Does not apply to numeric values.
 - Control printing of number:
 - digits The number of digits to the right of the decimal place.
 - scientific use TRUE for scientific notation, FALSE for standard notation

String Manipulation cont...

- Type the following commands:
 - `format(13.7, nsmall = 3)` # nsmall=number of decimal point
 - `format(c(6, 13.1), digits = 2)` #total number of digits (include decimal)
 - Justify:
 - `format(c("A", "BB", "CCC"), width = 5, justify = "centre")`
 - `format(c("A", "BB", "CCC"), width = 5, justify = "left")`
 - `format(c("A", "BB", "CCC"), width = 5, justify = "right")`
 - `format(123456789, big.mark = ",")`
 - Format a data frame:
 - `zz <- data.frame("(row names)"= c("aaaaa", "b"), check.names = FALSE)`
 - `format(zz)`
 - `format(zz, justify = "left")`
 - Use of scientific:
 - `format(2^31-1, scientific = TRUE)`

String Manipulation cont...

- `sprintf()` - a wrapper for the C function `sprintf()` that returns a formatted string combining text and variable values.

Function	Description
<code>%f</code>	Indicates 'fixed point' decimal notation
<code>%e</code> / <code>%E</code>	Exponential decimal notation
<code>%g</code>	No of significant digits (6 by default); eliminate superfluous zeros
<code>%s</code>	Character string
<code>%d</code>	Integer
<code>%x</code> / <code>%X</code>	hexadecimal

- Type the following commands:
 - `n<-c(2,3,5)`
 - `sprintf("%f", n[1])`
 - `sprintf("%.3f", n[2])` #3 decimal point
 - `sprintf("%1.0f", n[3])` #1 integer
 - `sprintf("%5.1f", n[1])`
 - `sprintf("%+f", n[1])` #with positive sign
 - `sprintf("%-10f", n[2])` # left justified
 - `sprintf("%s is %f feet tall\n", "Sven", 7.1)`
 - `sprintf("%s %d", "test", 1:3)`
 - `n <- 1:18`
 - `sprintf(paste0("e with %2d digits = %.", n, "g"), n, exp(1))`
 - `sprintf("%09s", month.name)` # pad with zero or spaces
 - `sprintf("min 10-char string '%10s'", c("a", "ABC", "and an even longer one"))`

String Manipulation cont...

- String manipulation function:

Function	Description
nchar()	number of characters
tolower()	convert to lower case
toupper()	convert to upper case
casefold()	case folding
chartr()	character translation (replacement)
abbreviate()	abbreviation
substring()	substrings of a character vector
substr()	substrings of a character vector
grep()	search for a pattern

String Manipulation cont...

- `casefold()` - a wrapper for `tolower` and `toupper`
 - `casefold(x, upper=FALSE)` #by default
 - Try the following commands:
 - `casefold("aLL ChaRacterS in LoweR caSe")`
 - `casefold("All ChaRacterS in Upper Case", upper = TRUE)`
- `chartr()` - to replace character
 - `chartr(old, new, x)`
 - Number of character for both old and new must be equal
 - Commands:
 - `chartr("ai", "X", "This is a bad example")` #error
 - `crazy = c("Heres to the crazy ones", "The misfits", "The rebels")`
 - `chartr("aei", "#!?", crazy)`

String Manipulation cont...

- `abbreviate()` – shorten the words/strings
 - `some_colors = colors()[1:4]`
 - `some_colors`
 - `colors1 = abbreviate(some_colors)`
 - `colors1`
 - `#abbreviate` with minimum length of char is 5
 - `colors2 = abbreviate(some_colors, minlength = 5)`
 - `colors3 = abbreviate(some_colors, minlength = 3, method = "both.sides")` `#abbreviate` from both site

String Manipulation cont...

- Count the number of character
 - `x <- c("asfef", "qwerty", "yuiop[", "b", "stuff.blah.yech")`
 - `nchar(x)`
- `substring()`
 - `substr(x, 2, 5)` #start at character 2, stop at 5
 - `substring(x, 2, 4:6)` #first element is 2, last element is 4 till 6
- `strsplit()` – split the elements of character vector x into substrings based on another string split in x
 - `strsplit(x, "e")` #split string x based on letter 'e'
 - `strsplit("6-16-2018",split="-")`

String Manipulation cont...

- A sample function which reverse the string
 - `strReverse <- function(x){`
 `sapply(lapply(strsplit(x, NULL), rev), paste, collapse = "")}`
- Example input of x:
 - `x = "abc"`
 - `strsplit(x, NULL)` will produce "a" "b" "c" and `lapply` will apply `rev` to all x elements and a reverse order is produced: "c" "b" "a"
 - `sapply` apply the `paste` to combine all the x element to become one string
 - With `lapply` and `sapply`, you can parse more than one element:
 - `strReverse(c("abc", "Statistics"))`

String Manipulation cont...

- Pattern matching
 - `grep()` – search a specified substring pattern in a vector string and return the match pattern in the form of index
 - `grep("Pole", c("Equator", "North Pole", "South Pole"))`
 - #match either e or i
 - `grep(pattern = "[ei]", c("Equator", "North Pole", "South Pole"), value = TRUE)`
 - `text = c("one word", "a sentence", "you and me", "three two one")`
 - `pat = "one"`
 - `grep(pat, text, value = TRUE)` # show matched text
 - `regexpr()` – finds the match character position within a text
 - `regexpr("uat", "Equator")`
 - `x = regexpr(pat, text)`
 - `substring(text, x, x + attr(x, "match.length") - 1)`
 - `gregexpr()` – same like `regexpr` but it finds all instances of patterns
 - `gregexpr("iss", "Mississippi")`
 - “iss” appear twice in character position 2 and 5
 - `(.)` represents any single element
 - `grep("o.e", c("Equator", "North Pole", "South Pole"))`
 - `grep("N..t", c("Equator", "North Pole", "South Pole"))`
 - Use with quantifier
 - `people = c("rori", "emilia", "matteo", "mehmet", "filipe", "anna", "tyler", "rasmus", "jacob", "youna", "flora", "adi")`
 - `grep(pattern = "m?", people, value = TRUE)` # match m at most once
 - `grep(pattern = "m{1}", people, value = TRUE, perl = FALSE)` #match m exactly one
 - In case, the match character is a `(.)` then we need to use backslash to escape the metacharacter
 - `grep(".", c("Equa.tor", "North Pole", "South Pole"))`
 - `grep("\\.", c("Equa.tor", "North Pole", "South Pole"))`

String Manipulation cont...

- Pattern Matching
 - Table of Pattern matching

Anchor	Description
[aeiou]	match any one lower case vowel
[AEIOU]	match any one upper case vowel
[0123456789]	match any digit
[0-9]	match any digit (same as previous class)
[a-z]	match any lower case ASCII letter
[A-Z]	match any upper case ASCII letter
[a-zA-Z0-9]	match any of the above classes
[^aeiou]	match anything other than a lowercase vowel
[^0-9]	match anything other than a digit

- Table of Quantifier

Quantifier	Description
?	The preceding item is optional and will be matched at most once
*	The preceding item will be matched zero or more times
+	The preceding item will be matched one or more times
{n}	The preceding item is matched exactly n times
{n,}	The preceding item is matched n or more times
{n,m}	The preceding item is matched at least n times, but not more than m times

String Manipulation cont...

- Replacement though gsub()
 - Table of POSIX characters

Class	Description
<code>[:lower:]</code>	Lower-case letters
<code>[:upper:]</code>	Upper-case letters
<code>[:alpha:]</code>	Alphabetic characters (<code>[:lower:]</code> and <code>[:upper:]</code>)
<code>[:digit:]</code>	Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
<code>[:alnum:]</code>	Alphanumeric characters (<code>[:alpha:]</code> and <code>[:digit:]</code>)
<code>[:blank:]</code>	Blank characters: space and tab
<code>[:cntrl:]</code>	Control characters
<code>[:punct:]</code>	Punctuation characters: ! " # % & ' () * + , - . / : ;
<code>[:space:]</code>	Space characters: tab, newline, vertical tab, form feed, carriage return, and space
<code>[:xdigit:]</code>	Hexadecimal digits: 0-9 A B C D E F a b c d e f
<code>[:print:]</code>	Printable characters (<code>[:alpha:]</code> , <code>[:punct:]</code> and space)
<code>[:graph:]</code>	Graphical characters (<code>[:alpha:]</code> and <code>[:punct:]</code>)

- `la_vie = "La vie en #FFC0CB (rose);\nCest la vie! \ttres jolie "`
- `cat(la_vie)`
- `gsub(pattern = "[:blank:]", replacement = "", la_vie) # remove space`
- `gsub(pattern = "[:xdigit:]", replacement = "", la_vie) #remove hexadig`

String Manipulation cont...

- `paste("University", "of", "California", "Berkeley", NULL, character(0), "Go Bears!")`
 - `paste()` cannot remove `NULL` or `character(0)`
 - That is why `stringr` is here, to manage and process string in a more consistent way

Exercise 3.5 – 25 minutes

- Let us create a customize print function by follow the steps here:
 - Assign a list which contains name = Joe, salary = 55000 and union = T to variable j
 - Assign “employee” to class(j)
 - Write a print function call print.employee:
 - This function receive a parameter
 - It can print out the following output [Hints: cat is useful]:
 - Joe
 - Salary 55000
 - Union member TRUE
- ```
j <- list(name="Joe", salary=55000, union=T)
create a function
print.employee <- function(k) {
 cat(k$name, "\n")
 cat("salary", k$salary, "\n")
 cat("union member", k$union, "\n")
}
```