Generating Data and Manipulating Objects (Part1)

The workshop of Generating data and manipulating objects will include two parts. For the first part, we will introduce data object, structure, some useful functions to describe data, and data manipulation. For the second part, we will learn manipulating data using the package dplyr. If there is any questions, please feel free to reach out to me (jingwen.gu@nih.gov) or Poorani (poorani.subramanian@nih.gov).

0.1 Create workspace

- create/close project
- save script

0.2 Set working directory

- set work directory: setwd("file_path") or setwd(file.choose()) to manually select a path
- get work directory: getwd()
- use R project: path will be set to the folder of the project file.

```
getwd()
```

0.3 install R packages

- install by command

```
install.packages("readr")  #install one package
install.packages(c("dplyr", "Stat2Data",
    "naniar", "tidyverse"))  # install multiple packages
```

- install in Packages panel
- manual install

Notes:
1. You only need to install the package once, call it (library the package) everytime when you want to use it.
2. When you want to make a comment, use "#", everything after "#"" in the line will not be executed.

```
library(Stat2Data)  # library package before use it
library(dplyr)
library(tidyverse)
```

0.4 Operator Five types of operator:1) Arithmetic operators 2) Relational Operators 3) Logical Operators 4) Assignment Operators 5) Miscellaneous Operators

Type	Operator	Description
1	+,-,*,/,^	Addition, subtraction, multiplication,division, exponent
2	<	Less than
2	>	Greater than
2	==	Equal to
2	!=	Not equal to
2	>=	Greater than or equal to
2	<=	Less than or equal to
3	!	logical NOT
3	&	Element-wise logical AND
3	&&	Logical AND
3	|	Element-wise logical OR
3	||	Logical OR
4	<-, =	Left assignment
4	->	Right assignment
5	$	List subset
5	%in%	Matching operator

- see the full list from R Operators

0.5 data type and generate data object

- Types of R object
- merge

Function	Description
c	Create vector
data.frame	Create data frame
as.matrix	Transform into matrix
list	Create list
tibble	Create tibble data frame

```
# create a vector
id <- c(seq(1, 5, by = 1))
name <- c("apple", "pear", "strawberry",
    "banana", "watermelon")

# create data frame
fruit_menu <- data.frame(id, name)
fruit_menu
```

```
##   id       name
## 1  1      apple
## 2  2       pear
## 3  3 strawberry
## 4  4     banana
## 5  5 watermelon
```

```r
# create matrix
fruit_matrix <- as.matrix(fruit_menu)
fruit_matrix
```

```
##      id  name
## [1,] "1" "apple"
## [2,] "2" "pear"
## [3,] "3" "strawberry"
## [4,] "4" "banana"
## [5,] "5" "watermelon"
```

```r
num_mat <- matrix(c(1:9), ncol = 3)

# create list
fruit_list <- list(fruit_menu)
fruit_list
```

```
## [[1]]
##   id       name
## 1  1      apple
## 2  2       pear
## 3  3 strawberry
## 4  4     banana
## 5  5 watermelon
```

```r
# create another data frame for merging
# practice
fruit_price <- data.frame(id = c(1, 2, 3,
    5, 6), price = c(3, 2, 4, 2, 10))
fruit_price
```

```
##   id price
## 1  1     3
## 2  2     2
## 3  3     4
## 4  5     2
## 5  6    10
```

```r
# merge two data sets
complete_menu_data <- merge(fruit_menu, fruit_price,
    all.x = T, by.x = "id", by.y = "id")
complete_menu_data  # all rows in fruit_menu
```

```
##   id       name price
## 1  1      apple     3
## 2  2       pear     2
## 3  3 strawberry     4
## 4  4     banana    NA
## 5  5 watermelon     2
```

```r
complete_price_data <- merge(fruit_menu,
    fruit_price, all.y = T, by.x = "id",
    by.y = "id")
complete_price_data  # all rows in fruit_price
```

```
##   id       name price
## 1  1      apple     3
## 2  2       pear     2
## 3  3 strawberry     4
## 4  5 watermelon     2
## 5  6       <NA>    10
```

0.6 import data

- use command (depend on file type)
- there are many other packages could help read in different types of files, such as "readxl" or others packages listed

```r
library(readr)
BirdNest <- read_csv("BirdNest.csv")  # read data from csv file
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Species = col_character(),
##   Common = col_character(),
##   Page = col_double(),
##   Length = col_double(),
##   Nesttype = col_character(),
##   Location = col_character(),
##   No.eggs = col_double(),
##   Color = col_double(),
##   Incubate = col_double(),
##   Nestling = col_double(),
##   Totcare = col_double(),
##   Closed. = col_double()
## )
```

```r
# If you did not set work directory or
# would like to read in file from other
# folder instead of working directory,
# specify path above
```

- import from "Files" panel
- import using file.choose()
- import build-in R data

```r
library(Stat2Data)
data(BirdNest)  # get BirdNest data
str(BirdNest)
```

```
## 'data.frame':    84 obs. of  12 variables:
##  $ Species : Factor w/ 84 levels "Agelaius phoeniceus",..: 81 45 42 43 41 65 24 61 48 2 ...
##  $ Common  : Factor w/ 84 levels "American Dipper",..: 29 71 5 16 27 30 35 76 62 31 ...
##  $ Page    : int  360 368 372 372 374 378 382 382 394 394 ...
##  $ Length  : num  20 20 20 22.5 17 17 15 15 16 18.5 ...
##  $ Nesttype: Factor w/ 7 levels "burrow","cavity",..: 4 2 2 2 2 4 4 4 7 6 ...
##  $ Location: Factor w/ 9 levels "bank","bridge",..: 6 6 6 6 6 2 8 6 6 7 ...
##  $ No.eggs : num  3.5 3.5 4.5 4.5 4.5 4.5 3.5 3 5 3.5 ...
##  $ Color   : int  1 1 1 1 1 0 0 1 1 1 ...
##  $ Incubate: num  17 15.5 15 14 14 16 14 14.5 NA 11.5 ...
##  $ Nestling: num  17 17 15 16.5 14 15.5 16 15 19 9.5 ...
##  $ Totcare : num  34 32.5 30 30.5 28 31.5 30 29.5 NA 21 ...
##  $ Closed. : int  0 1 1 1 1 0 0 0 1 0 ...
```

```
help(read_csv)
```

0.7 getting help

Introduction to BirdNest

- Nest and species characteristics for North American passerines
- Amy R. Moore, as a student at Grinnell College in 1999, wanted to study the relationship between species characteristics and the type of nest a bird builds, using data collected from available sources. For the study, she collected data by species for 84 separate species of North American passerines. Source

0.8 basic functions to learn about your data

Funtion	Description
`summary()`	Describe data
`sapply(data, class)`	See all variable type
`dim()`	Dimension of data
`colnames()`	Column names
`rownames()`	Row names
`complete.cases()`	Filter on complete cases
`class()`	Data type
`head()`	first six rows of the data, or with number specified, "n=10"
`!`	Indicates logical negation
`table()`	Frequency table
`unique()`	See all unique values
`length()`	Report length of vector
`is.na()`	Check for missing values
`rbind()`	Row bind
`cbind()`	Column bind

```r
summary(BirdNest)  # describe data
sapply(BirdNest, class)  # check the type of each variable in BirdNest
dim(BirdNest)  # dimension
colnames(BirdNest)
BirdNest[!complete.cases(BirdNest), ]  # return rows with missing value in BirdNest
class(BirdNest)  # the type of the data set
head(BirdNest)  # first six rows
table(BirdNest$Location)  # return all unique value with frequency in Location variable of BirdNest
unique(BirdNest$Location)  # return all unique value in factor
length(unique(BirdNest$Location))  # return the number of unique value in factor
table(is.na(BirdNest$Location))
table(is.na(BirdNest$Totcare))
```

0.9 creating new variable and select variables by critieria

- Create a new variable by patterns from other variables. Other useful function are: grepl, gusb. Similar formats could be used to select variables from column names.

```r
BirdNest$cup_type <- ifelse(BirdNest$Nesttype ==
    "cup", 1, 0)
table(BirdNest$cup_type)  # create binary variable to indicate 'cup' type nest
```

```
##
##  0  1
## 31 53
```

```r
BirdNest$white_cup <- ifelse(BirdNest$Nesttype ==
    "cup" & BirdNest$Color == 1, 1, 0)
table(BirdNest$white_cup)  # create binary variable to indicate 'cup' and 'white' nest
```

```
##
##  0  1
## 40 44
```

```r
BirdNest$cup_cavity <- ifelse(BirdNest$Nesttype %in%
    c("cup", "cavity"), 1, 0)
table(BirdNest$cup_cavity)  # create binary variable to indicate 'cup' or 'cavity' type nest
```

```
##
##  0  1
## 14 70
```

```r
BirdNest$cu_type <- ifelse(grepl("cu", BirdNest$Nesttype) ==
    TRUE, 1, 0)
table(BirdNest$cu_type)  # create binary variable to indicate 'cu' type nest
```

```
##
##  0  1
## 31 53
```

```
select1 <- BirdNest[, grepl("Co", colnames(BirdNest))]
colnames(select1)
```

```
## [1] "Common" "Color"
```

```
select2 <- BirdNest[, colnames(BirdNest) %in%
    c("Common", "Color")]
colnames(select2)
```

```
## [1] "Common" "Color"
```

0.10 missing value In real analysis, missing values could be defined in different ways. We could collect the expression of missing value in the data and clean it by standard expression.

```
example <- data.frame(id = seq(1:4), name = c("Apple",
    "NA", "Orange", "Banana"))
example
```

```
##   id   name
## 1  1  Apple
## 2  2     NA
## 3  3 Orange
## 4  4 Banana
```

```
library(naniar)
na_strings <- c("NA")
example_clean <- example %>% replace_with_na_all(condition = ~.x %in%
    na_strings)

example_clean
```

```
## # A tibble: 4 x 2
##      id name
##   <int> <chr>
## 1     1 Apple
## 2     2 <NA>
## 3     3 Orange
## 4     4 Banana
```

0.11 Transform long and wide data (packages: tidyr and reshape2)
From long to wide:spread,dcast; from wide to long:gather, melt

```
long_data <- read.table(header = TRUE, text = "
 subject sex condition measurement
       1   M   control         7.9
       1   M     cond1        12.3
       1   M     cond2        10.7
       2   F   control         6.3
       2   F     cond1        10.6
```

```
       2   F     cond2        11.1
       3   F   control         9.5
       3   F     cond1        13.1
       3   F     cond2        13.8
       4   M   control        11.5
       4   M     cond1        13.4
       4   M     cond2        12.9
")

# long to wide
library(tidyr)
wide_data <- spread(long_data, condition,
    measurement)  # tidyr
wide_data
```

```
##   subject sex cond1 cond2 control
## 1       1   M  12.3  10.7     7.9
## 2       2   F  10.6  11.1     6.3
## 3       3   F  13.1  13.8     9.5
## 4       4   M  13.4  12.9    11.5
```

```
# long to wide
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
wide_data2 <- dcast(long_data, subject +
    sex ~ condition, value.var = "measurement")  # reshape2
wide_data2
```

```
##   subject sex cond1 cond2 control
## 1       1   M  12.3  10.7     7.9
## 2       2   F  10.6  11.1     6.3
## 3       3   F  13.1  13.8     9.5
## 4       4   M  13.4  12.9    11.5
```

```
# wide to long
data_long <- gather(wide_data, condition,
    measurement, cond1:control, factor_key = TRUE)  # tidyr
data_long
```

```
##    subject sex condition measurement
## 1        1   M     cond1        12.3
## 2        2   F     cond1        10.6
## 3        3   F     cond1        13.1
## 4        4   M     cond1        13.4
```

```
## 5        1   M     cond2        10.7
## 6        2   F     cond2        11.1
## 7        3   F     cond2        13.8
## 8        4   M     cond2        12.9
## 9        1   M   control         7.9
## 10       2   F   control         6.3
## 11       3   F   control         9.5
## 12       4   M   control        11.5
```

```r
# wide to long
data_long2 <- melt(wide_data, id.vars = c("subject",
    "sex"))  # reshape2
data_long2
```

```
##    subject sex variable value
## 1        1   M    cond1  12.3
## 2        2   F    cond1  10.6
## 3        3   F    cond1  13.1
## 4        4   M    cond1  13.4
## 5        1   M    cond2  10.7
## 6        2   F    cond2  11.1
## 7        3   F    cond2  13.8
## 8        4   M    cond2  12.9
## 9        1   M  control   7.9
## 10       2   F  control   6.3
## 11       3   F  control   9.5
## 12       4   M  control  11.5
```

Example from R cookbook

In the end, the function above are handy and good to know when looking into the data. In practice, it is also good to maintain the manipulation in a tidy format. In the second part, we will introduce use pipe format and manipulate data using package dplyr.