

# R basics

SIT22003: Introduction to Data Science  
Presented by Hyebyong Choi

# Conditionals - If Statement

```
if (condition) {  
  expr  
}
```

```
# Variables related to your last day of recordings  
medium <- "LinkedIn"  
num_views <- 14
```

```
      true  
if (medium == "LinkedIn") {  
  print("Showing LinkedIn information")  
}
```

```
## [1] "Showing LinkedIn information"
```

```
      false  
if (num_views > 15) {  
  print("You're popular!")  
}
```

No output

# Conditionals - If Statement

*# Variables related to your last day of recordings*

```
medium <- "LinkedIn"
```

```
num_views <- 14
```

**true**

```
if (medium == "LinkedIn") {  
  print("Showing LinkedIn information") <-  
} else {  
  print("Unknown medium")  
}
```

```
## [1] "Showing LinkedIn information"
```

```
if (num_views > 15) {  
  print("You're popular!")  
} else {  
  print("Try to be more visible!") <-  
}
```

```
## [1] "Try to be more visible!"
```

```
if (condition) {  
  expr1  
} else {  
  expr2  
}
```

# For Loop

```
for(var in seq) {  
    expr  
}
```

```
cities <- c("New York", "Paris",  
           "London", "Tokyo",  
           "Rio de Janeiro", "Cape  
Town")  
for(city in cities) {  
    print(city)  
}  
  
## [1] "New York"  
## [1] "Paris"  
## [1] "London"  
## [1] "Tokyo"  
## [1] "Rio de Janeiro"  
## [1] "Cape Town"
```

# Vectorized Operation

```
numbers_vector
```

```
## [1] 1 3 4 2 6 8 7 5
```

```
numbers_even_odd
```

```
## [1] "odd" "odd" "even" "even" "even" "even" "odd" "odd"
```

How could you **numbers\_even\_odd** vector from **numbers\_vector**?

Maybe for-loop with if command?

**Not a good idea in R**

# Vectorized Operation

```
numbers_vector <- c(1,3,4,2,6,8,7,5)

numbers_even_odd <- ifelse(numbers_vector %% 2 == 0, 'even', 'odd')

numbers_even_odd
## [1] "odd"  "odd"  "even" "even" "even" "even" "odd"  "odd"
table(numbers_even_odd)
## numbers_even_odd
## even  odd
##    4    4
```

**Easier! and Much More Efficient!**

# Vectorized Operation

## Adding New Variable “Fuel\_efficiency” to mtcars

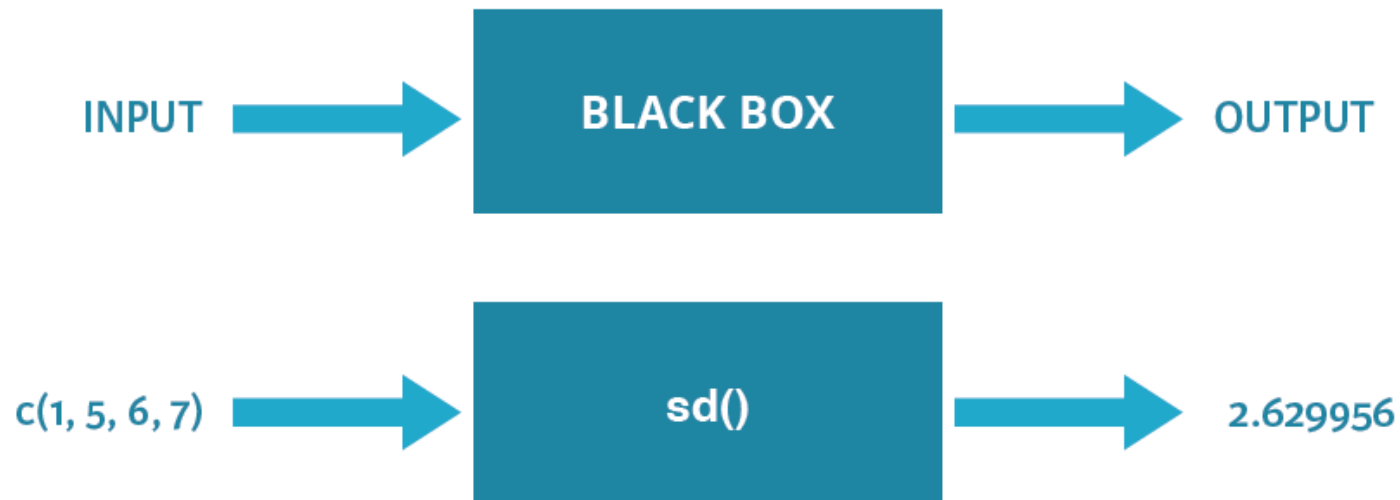
```
avg_mpg <- mean(mtcars$mpg)
new_var <- ifelse(mtcars$mpg >= avg_mpg, 'good', 'bad')
```

```
## adding new variable to mtcars
mtcars$fuel_efficiency <- new_var
head(mtcars)
```

[illegible]

# Function

- Function is an automatized routine of task
  - Print, mean, max, min are all function





# User-defined Functions

```
cube <- function(n) {  
  return(n*n*n)  
}
```

```
cube(10)
```

```
## [1] 1000
```

```
cube(1:5)
```

```
## [1] 1 8 27 64 125
```

# User-defined Functions

```
is.even.number <- function(n) {  
  n %% 2 == 0  
}
```

```
is.even.number(10)
```

```
## [1] TRUE
```

```
is.even.number(5)
```

```
## [1] FALSE
```

```
is.even.number(c(1,2,5,6,7,9,15))
```

```
## [1] FALSE TRUE FALSE TRUE FALSE FALSE FALSE
```

```
numbers_vector <- c(1,3,4,2,6,8,7,5)
```

```
ifelse(is.even.number(numbers_vector), 'even', 'odd')
```

```
## [1] "odd" "odd" "even" "even" "even" "even" "odd" "odd"
```

# User-defined Functions

```
is.even.number <- function(n) {  
  n %% 2 == 0  
}
```

```
is.even.number(10)
```

```
## [1] TRUE
```

```
is.even.number(5)
```

```
## [1] FALSE
```

```
is.even.number(c(1,2,5,6,7,9,15))
```

```
## [1] FALSE TRUE FALSE TRUE FALSE FALSE FALSE
```

```
numbers_vector <- c(1,3,4,2,6,8,7,5)
```

```
ifelse(is.even.number(numbers_vector), 'even', 'odd')
```

```
## [1] "odd" "odd" "even" "even" "even" "even" "odd" "odd"
```

# User-defined Functions

```
diff.max.min <- function(...) {  
  a <- c(...)  
  largest <- max(a)  
  smallest <- min(a)  
  
  largest - smallest  
}
```

```
diff.max.min(6,5,6,23,4,25)
```

```
## [1] 21
```

```
diff.max.min(-55, 100, 23, -7)
```

```
## [1] 155
```

# Vectorized operations

- Most of R operations can be applied to both a vector and scalar

```
my.vector <- c(1,3,5,8,13)
my.vector * 2
## [1]  2  6 10 16 26
my.vector >= 5
## [1] FALSE FALSE  TRUE  TRUE  TRUE
my.vector < 10
## [1]  TRUE  TRUE  TRUE  TRUE FALSE
my.vector >= 5 & my.vector < 10
## [1] FALSE FALSE  TRUE  TRUE FALSE
```

# Vectorized operations

- Most of R operations can be applied to both a vector and scalar

```
my.vector
## [1] 1 3 5 8 13
sum(my.vector)
## [1] 30
mean(my.vector)
## [1] 6
median(my.vector)
## [1] 5
min(my.vector)
## [1] 1
max(my.vector)
## [1] 13
```

**try not to use “for-loop”  
use vectorized operations instead!**

```
summary(my.vector)
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##         1         3         5         6         8        13

ifelse(my.vector %% 2 == 0, 'even', 'odd')
## [1] "odd"  "odd"  "odd"  "even" "odd"
```

# Importing Data in to R

- What to import?

- 5 Types

- Flat files



- Data from Excel



- Databases




- Web



- Statistical software



# Flat Files

 states.csv

Comma Separated Values

state,capital,pop\_mill,area\_sqm  
South Dakota,Pierre,0.853,77116  
New York,Albany,19.746,54555  
Oregon,Salem,3.970,98381  
Vermont,Montpelier,0.627,9616  
Hawaii,Honolulu,1.420,10931

← Field names



```
> wanted_df
```

|   | state        | capital    | pop_mill | area_sqm |
|---|--------------|------------|----------|----------|
| 1 | South Dakota | Pierre     | 0.853    | 77116    |
| 2 | New York     | Albany     | 19.746   | 54555    |
| 3 | Oregon       | Salem      | 3.970    | 98381    |
| 4 | Vermont      | Montpelier | 0.627    | 9616     |
| 5 | Hawaii       | Honolulu   | 1.420    | 10931    |



# read.csv

The `utils` package, which is automatically loaded in your R session on startup, can import CSV files with the `read.csv()` function.

“`swimming_pools.csv`” contains data on swimming pools in Brisbane, Australia (Source: **data.gov.au**). The file contains the column names in the first row. It uses a comma to separate values within rows.

```
# Import swimming_pools.csv: pools
pools <- read.csv('swimming_pools.csv')

# Print the structure of pools
str(pools)

## 'data.frame':    20 obs. of  4 variables:
##  $ Name      : Factor w/ 20 levels "Acacia Ridge Leisure Centre",...: 1 2 3 4 5 6 19
##  $ Address   : Factor w/ 20 levels "1 Fairlead Crescent, Manly",...: 5 20 18 10 9 11
##  $ Latitude  : num  -27.6 -27.6 -27.6 -27.5 -27.4 ...
##  $ Longitude : num  153 153 153 153 153 ...
```

# read.csv stringsAsFactor

With `stringsAsFactors`, you can tell R whether it should convert strings in the flat file to factors.

```
# Import swimming_pools.csv correctly: pools
pools <- read.csv('swimming_pools.csv', stringsAsFactors = FALSE)

# Check the structure of pools
str(pools)

## 'data.frame':    20 obs. of  4 variables:
##  $ Name      : chr  "Acacia Ridge Leisure Centre" "Bellbowrie Pool" "Carole Park"
##              "Centenary Pool (inner City)" ...
##  $ Address   : chr  "1391 Beaudesert Road, Acacia Ridge" "Sugarwood Street, Bellbowrie"
##              "Cnr Boundary Road and Waterford Road Wacol" "400 Gregory Terrace, Spring Hill" ...
##  $ Latitude  : num  -27.6 -27.6 -27.6 -27.5 -27.4 ...
##  $ Longitude: num  153 153 153 153 153 ...
```

# Converting Types Afterward

```
# Import swimming_pools.csv: pools
pools <- read.csv('swimming_pools.csv')

pools$Name <- as.character(pools$Name)
pools$Address <- as.character(pools$Address)

# Print the structure of pools
str(pools)

## 'data.frame':    20 obs. of  4 variables:
##  $ Name      : chr  "Acacia Ridge Leisure Centre" "Bellbowrie Pool" "Carole Park"
##              "Centenary Pool (inner City)" ...
##  $ Address   : chr  "1391 Beaudesert Road, Acacia Ridge" "Sugarwood Street,
##              Bellbowrie" "Cnr Boundary Road and Waterford Road Wacol" "400 Gregory Terrace, Spring
##              Hill" ...
##  $ Latitude  : num  -27.6 -27.6 -27.6 -27.5 -27.4 ...
##  $ Longitude: num  153 153 153 153 153 ...
```

# read.table

- Read any tabular file as a data frame
- Number of arguments is huge

 states2.txt

```
state/capital/pop_mill/area_sqm  
South Dakota/Pierre/0.853/77116  
New York/Albany/19.746/54555  
Oregon/Salem/3.970/98381  
Vermont/Montpelier/0.627/9616  
Hawaii/Honolulu/1.420/10931
```

```
> read.table("states2.txt",  
             header = TRUE,    first row lists variable names (default FALSE)  
             sep = "/",        field separator is a forward slash  
             stringsAsFactors = FALSE)
```

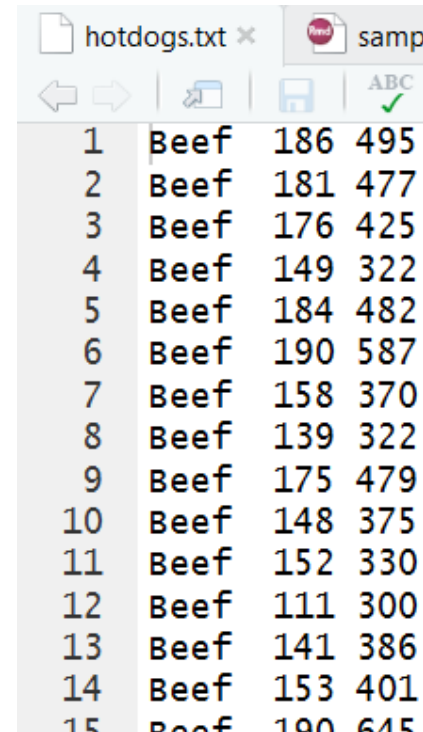
|   | state        | capital    | pop_mill | area_sqm |
|---|--------------|------------|----------|----------|
| 1 | South Dakota | Pierre     | 0.853    | 77116    |
| 2 | New York     | Albany     | 19.746   | 54555    |
| 3 | Oregon       | Salem      | 3.970    | 98381    |
| 4 | Vermont      | Montpelier | 0.627    | 9616     |
| 5 | Hawaii       | Honolulu   | 1.420    | 10931    |

# reading hotdog.txt

[hotdogs.txt](#), contains information on sodium and calorie levels in different hotdogs (Source: **UCLA**).

The dataset has 3 variables, but the variable names are not available in the first line of the file.

The file uses [tabs](#) as field [separators](#).



|    |      |     |     |
|----|------|-----|-----|
| 1  | Beef | 186 | 495 |
| 2  | Beef | 181 | 477 |
| 3  | Beef | 176 | 425 |
| 4  | Beef | 149 | 322 |
| 5  | Beef | 184 | 482 |
| 6  | Beef | 190 | 587 |
| 7  | Beef | 158 | 370 |
| 8  | Beef | 139 | 322 |
| 9  | Beef | 175 | 479 |
| 10 | Beef | 148 | 375 |
| 11 | Beef | 152 | 330 |
| 12 | Beef | 111 | 300 |
| 13 | Beef | 141 | 386 |
| 14 | Beef | 153 | 401 |
| 15 | Beef | 190 | 615 |

# reading hotdog.txt

```
hotdogs <- read.table('hotdogs.txt',  
                      sep = '\\t')
```

```
# Call head() on hotdogs  
head(hotdogs)
```

```
##      V1  V2  V3  
## 1 Beef 186 495  
## 2 Beef 181 477  
## 3 Beef 176 425  
## 4 Beef 149 322  
## 5 Beef 184 482  
## 6 Beef 190 587
```

# reading hotdog.txt

```
hotdogs <- read.table('hotdogs.txt',  
                      sep = '\\t',  
                      col.names = c("type", "calories", "sodium"))
```

*# Call head() on hotdogs*

```
head(hotdogs)
```

| ##   | type | calories | sodium |
|------|------|----------|--------|
| ## 1 | Beef | 186      | 495    |
| ## 2 | Beef | 181      | 477    |
| ## 3 | Beef | 176      | 425    |
| ## 4 | Beef | 149      | 322    |
| ## 5 | Beef | 184      | 482    |
| ## 6 | Beef | 190      | 587    |

# Exporting Data Frame as csv file

write.csv function

*# add new variable named "cal.type"*

```
hotdogs$cal.type <- ifelse(hotdogs$calories >= 150, 'heavy',  
'light')
```

```
head(hotdogs)
```

| ## |   | type | calories | sodium | cal.type |
|----|---|------|----------|--------|----------|
| ## | 1 | Beef | 186      | 495    | heavy    |
| ## | 2 | Beef | 181      | 477    | heavy    |
| ## | 3 | Beef | 176      | 425    | heavy    |
| ## | 4 | Beef | 149      | 322    | light    |
| ## | 5 | Beef | 184      | 482    | heavy    |
| ## | 6 | Beef | 190      | 587    | heavy    |

row names

```
write.csv(hotdogs, "newhotdog.csv", row.names = F)
```



# Exporting Data Frame as tsv file

`write.table` function

```
write.table(hotdogs, "newhotdog.tsv",  
            row.names = F, sep = '\t')
```

| "type" | "calories" | "sodium" | "cal.type" |
|--------|------------|----------|------------|
| "Beef" | 186        | 495      | "heavy"    |
| "Beef" | 181        | 477      | "heavy"    |
| "Beef" | 176        | 425      | "heavy"    |
| "Beef" | 149        | 322      | "light"    |
| "Beef" | 184        | 482      | "heavy"    |
| "Beef" | 190        | 587      | "heavy"    |
| "Beef" | 158        | 370      | "heavy"    |
| "Beef" | 139        | 322      | "light"    |
| "Beef" | 175        | 479      | "heavy"    |
| "Beef" | 148        | 375      | "light"    |
| "Beef" | 152        | 330      | "heavy"    |
| "Beef" | 111        | 300      | "light"    |
| "Beef" | 141        | 386      | "light"    |
| "Beef" | 153        | 401      | "heavy"    |
| "Beef" | 190        | 645      | "heavy"    |
| "Beef" | 157        | 440      | "heavy"    |
| "Beef" | 131        | 317      | "light"    |
| "Beef" | 149        | 319      | "light"    |
| "Beef" | 135        | 298      | "light"    |
| "Beef" | 132        | 253      | "light"    |

# Save Your Variables as a File

- Any type of R variables can be saved in RData file
- `save(var1, var2, ..., file = "myfile.RData")`

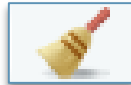
```
my.var <- 10
my.var2 <- c(1,4,6,22,3)
my.var3 <- c('John', 'Bob', 'Alice')
my.var4 <- data.frame(A = 1:3, B = 9:11)

save(my.var, my.var2, my.var3, my.var4,
      file = "myVariables.RData")
```

# Clear Your Workspace

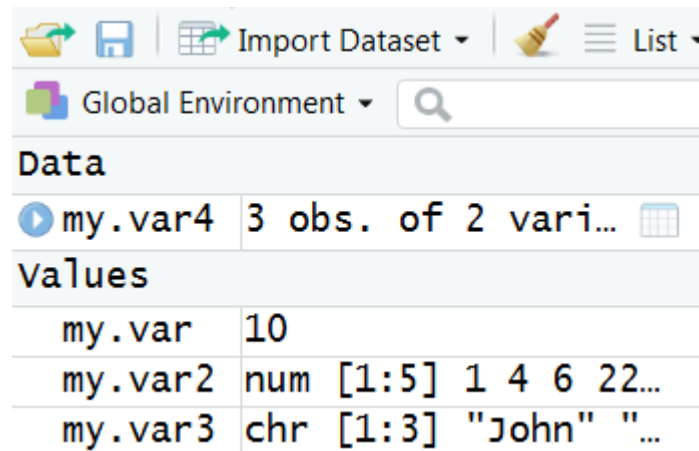
```
rm(list = ls())
```

or press sweep button



# Load Your Variables Back

```
load("myVariables.RData")
```



The screenshot shows the RStudio Global Environment pane. At the top, there are icons for file operations and a search bar. Below the search bar, the 'Data' section lists 'my.var4' with 3 observations and 2 variables. The 'Values' section shows the following data:

| Variable | Value                  |
|----------|------------------------|
| my.var   | 10                     |
| my.var2  | num [1:5] 1 4 6 22...  |
| my.var3  | chr [1:3] "John" "..." |

With Rdata files,

You may share your work in R with your colleges or

You may continue to work on the same data on different PC or

You may restore your work from some unexpected errors, etc.

# Saving working environment as a file

- To save everything in your working environment

```
save(list = ls(), file = "myWork.RData")
```

or press disk button



# The `apply()` Family

- The `apply()` family pertains to the R base package
- It applies functions to manipulate slices of data from `matrices`, `arrays`, `lists` and `dataframes`
- `apply()`, `lapply()` , `sapply()`, `vapply()`, `mapply()`, `rapply()`, and `tapply()`

For More Information:



<https://www.datacamp.com/community/tutorials/r-tutorial-apply-family>

# apply()

**apply(X, MARGIN, FUN, ...)**

- **X** is matrix or dataframe
- **MARGIN** is a variable defining how the function is applied:
  - MARGIN=1, it applies over rows
  - MARGIN=2, it works over columns
- **FUN** is the function that you want to apply to the data

# apply()

| X  |  | apply(X ,2, sum)   |            |            |             |            |            |
|--|--|--|------------|------------|-------------|------------|------------|
| Dimension — 2  |  |  |            |            |             |            |            |
| 1<br> |  | -1.7189391   | -1.0863995 | 1.0996117  | -0.55559727 | -0.1792310 | -0.8088577 |
|  |  | -2.2542126   | -1.3201873 | -2.0533779 | 1.29055209  | 0.3264156  | 0.5412132  |
|  |  | 1.9874737  | 0.6265486  | -0.3684977 | 1.40028967  | -0.7574303 | -2.3241569 |
|  |  | 0.2140376  | 0.8850445  | 1.4782993  | -1.28177703 | -0.5015628 | 1.1537703  |
|  |  | 0.9637687  | 1.3191502  | 0.8000988  | 0.09345943  | 1.4535431  | 1.0935720  |
|  |  |  |            |            |             |            |            |
| Result   |  | sum  |            |            |             |            |            |
|  |  | -0.8078717   | 0.4241565  | 0.9561342  | 0.9469269   | 0.3417346  | -0.3444591 |



# apply()

```
set.seed(2018)
myMat <- matrix(runif(12), ncol = 4)
myMat

##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.33615347 0.1974336 0.6067589 0.5468495
## [2,] 0.46372327 0.4743142 0.1300121 0.3956160
## [3,] 0.06058539 0.3010486 0.9586547 0.6645386

apply(myMat, 1, mean)

## [1] 0.4217989 0.3659164 0.4962068

apply(myMat, 2, mean)

## [1] 0.2868207 0.3242655 0.5651419 0.5356680
```

# apply()

alternatively, colSums, rowSums, ...

```
head(iris)
```

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

```
apply(iris[, 1:4], 2, mean)
```

```
## Sepal.Length Sepal.Width Petal.Length Petal.Width
##      5.843333      3.057333      3.758000      1.199333
```

```
apply(iris[, 1:4], 1, mean)
```

```
## [1] 2.550 2.375 2.350 2.350 2.550 2.850 2.425 2.525 2.225 2.400 2.700
## [12] 2.500 2.325 2.125 2.800 3.000 2.750 2.575 2.875 2.675 2.675 2.675
.....
```

```
colMeans(iris[, 1:4])
```

```
## Sepal.Length Sepal.Width Petal.Length Petal.Width
##      5.843333      3.057333      3.758000      1.199333
```

# lapply()

- It applies function to dataframes, **lists** or vectors
- It gives you back a **list**

```
myList <- list(num = 3.14, chr = "char", logi = TRUE)
myList
```

```
## $num
## [1] 3.14
##
## $chr
## [1] "char"
##
## $logi
## [1] TRUE
```

```
lapply(myList, typeof)
```

```
## $num
## [1] "double"
##
## $chr
## [1] "character"
##
## $logi
## [1] "logical"
```

# lapply()

```
myList2 <- list(vec = 1:5, mat = matrix(runif(12), ncol = 4), df = iris)
result <- lapply(myList2, length)
result

## $vec
## [1] 5      # number of elements
##
## $mat
## [1] 12     # number of elements
##
## $df
## [1] 5      # number of columns
unlist(result) # list -> vector

## vec mat  df
##   5  12   5
```

# lapply()

```
lapply(c(1,4,9,16), sqrt)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
##
## [[4]]
## [1] 4
```

```
lapply(mtcars, max)
```

```
## $mpg
## [1] 33.9
##
## $cyl
## [1] 8
##
## $disp
## [1] 472
##
## $hp
## [1] 335
##
```

.....

```
unlist(lapply(mtcars, max))
```

```
##      mpg      cyl    disp      hp      drat      wt      qsec      vs      am
## 33.900    8.000 472.000 335.000    4.930    5.424   22.900    1.000    1.000
##      gear      carb
##    5.000    8.000
```

# sapply()

- It applies function to dataframes, lists or vectors
- It gives you back a **vector or matrix**

```
sapply(iris[, 1:4], mean)
```

```
## Sepal.Length Sepal.Width Petal.Length Petal.Width
##      5.843333      3.057333      3.758000      1.199333
```

```
sapply(iris, is.numeric)
```

```
## Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           TRUE           TRUE           TRUE           TRUE      FALSE
```

```
sapply(c(1,3,5,7,9), function(x) {x**2})
```

```
## [1]  1  9 25 49 81
```

# sapply()

```
myMat <- matrix(1:12, ncol = 4)
```

```
myMat
```

```
##      [,1] [,2] [,3] [,4]  
## [1,]    1    4    7   10  
## [2,]    2    5    8   11  
## [3,]    3    6    9   12
```

```
sapply(myMat, function(x) {x/2})
```

```
## [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0
```

```
sapply(pools, typeof)
```

```
##      Name      Address      Latitude      Longitude  
## "character" "character" "double"  "double"
```

# apply()

```
x <- apply(iris[,1:4], function(x) { x > 3 })  
head(x)
```

| ## |      | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width |
|----|------|--------------|-------------|--------------|-------------|
| ## | [1,] | TRUE         | TRUE        | FALSE        | FALSE       |
| ## | [2,] | TRUE         | FALSE       | FALSE        | FALSE       |
| ## | [3,] | TRUE         | TRUE        | FALSE        | FALSE       |
| ## | [4,] | TRUE         | TRUE        | FALSE        | FALSE       |
| ## | [5,] | TRUE         | TRUE        | FALSE        | FALSE       |
| ## | [6,] | TRUE         | TRUE        | FALSE        | FALSE       |

```
colSums(x)
```

| ## | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width |
|----|--------------|-------------|--------------|-------------|
| ## | 150          | 67          | 99           | 0           |



# tapply()

- **tapply(X, GRP\_VAR, FUN, ...)**
  - apply **FUN** to **X** after grouping with **GRP\_VAR**

```
tapply(iris$Sepal.Length, iris$Species, mean)
```

```
##      setosa versicolor  virginica  
##      5.006      5.936      6.588
```

```
tapply(mtcars$mpg, mtcars$cyl, mean)
```

```
##          4          6          8  
## 26.66364 19.74286 15.10000
```

```
tapply(mtcars$wt, mtcars$mpg>20, mean)
```

```
##      FALSE      TRUE  
## 3.838833 2.418071
```

# tapply()

- **tapply(X, GRP\_VAR, FUN, ...)**
  - apply **FUN** to **X** after grouping with **GRP\_VAR**

```
x <- tapply(mtcars$mpg, mtcars$cyl, function(x){x>20})
x

## $`4`
## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##
## $`6`
## [1] TRUE TRUE TRUE FALSE FALSE FALSE FALSE
##
## $`8`
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE

sapply(x, sum)

## 4 6 8
## 11 3 0
```

# aggregate()

- **aggregate(var1 ~ var2, data = X, FUN = func, ...)**
  - Apply **func** to **var1** of **X** after grouping by **var2**
  - Alternates to **tapply**
  - Result is **data.frame**

```
aggregate(mpg ~ cyl, data = mtcars, FUN = mean)
```

```
##      cyl      mpg
## 1     4 26.66364
## 2     6 19.74286
## 3     8 15.10000
```

```
aggregate(Sepal.Length ~ Species, data = iris, FUN = mean)
```

```
##      Species Sepal.Length
## 1    setosa      5.006
## 2 versicolor      5.936
## 3 virginica      6.588
```

# order() and sort()

- `order()` gives a vector of index of smallest element, second smallest, ..., the largest element
- `sort()` gives a sorted vector of numbers
- `decreasing` option to have result in descending order

```
my_vector <- c(6,12,4,89,23, 35)
order(my_vector)
```

```
## [1] 3 1 2 5 6 4
```

```
my_vector[order(my_vector)]
```

```
## [1] 4 6 12 23 35 89
```

```
sort(my_vector)
```

```
## [1] 4 6 12 23 35 89
```

```
my_vector[order(my_vector,decreasing = T)]
```

```
## [1] 89 35 23 12 6 4
```

```
sort(my_vector, decreasing = T)
```

```
## [1] 89 35 23 12 6 4
```

# Why Use `order()` instead of `sort()`

- Sorting dataframe

```
mtcars[order(mtcars$mpg, decreasing = T), ]
```

| ##                   | mpg  | cyl | disp  | hp  | drat | wt    | qsec  | vs | am | gear | carb |
|----------------------|------|-----|-------|-----|------|-------|-------|----|----|------|------|
| ## Toyota Corolla    | 33.9 | 4   | 71.1  | 65  | 4.22 | 1.835 | 19.90 | 1  | 1  | 4    | 1    |
| ## 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    |
| ## Lotus Europa      | 30.4 | 4   | 95.1  | 113 | 3.77 | 1.513 | 16.90 | 1  | 1  | 5    | 2    |
| ## Fiat X1-9         | 27.3 | 4   | 79.0  | 66  | 4.08 | 1.935 | 18.90 | 1  | 1  | 4    | 1    |
| ## Porsche 914-2     | 26.0 | 4   | 120.3 | 91  | 4.43 | 2.140 | 16.70 | 0  | 1  | 5    | 2    |
| ## Merc 240D         | 24.4 | 4   | 146.7 | 62  | 3.69 | 3.190 | 20.00 | 1  | 0  | 4    | 2    |
| ## Datsun 710        | 22.8 | 4   | 108.0 | 93  | 3.85 | 2.320 | 18.61 | 1  | 1  | 4    | 1    |
| ## Merc 230          | 22.8 | 4   | 140.8 | 95  | 3.92 | 3.150 | 22.90 | 1  | 0  | 4    | 2    |
| ## Toyota Corona     | 21.5 | 4   | 120.1 | 97  | 3.70 | 2.465 | 20.01 | 1  | 0  | 3    | 1    |
| ## Hornet 4 Drive    | 21.4 | 6   | 258.0 | 110 | 3.08 | 3.215 | 19.44 | 1  | 0  | 3    | 1    |
| ## Volvo 142E        | 21.4 | 4   | 121.0 | 109 | 4.11 | 2.780 | 18.60 | 1  | 1  | 4    | 2    |
| ## 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    |
| ## Ferrari Dino      | 19.7 | 6   | 145.0 | 175 | 3.62 | 2.770 | 15.50 | 0  | 1  | 5    | 6    |
| ## Merc 280          | 19.2 | 6   | 167.6 | 123 | 3.92 | 3.440 | 18.30 | 1  | 0  | 4    | 4    |
| ## Pontiac Firebird  | 19.2 | 8   | 400.0 | 175 | 3.08 | 3.845 | 17.05 | 0  | 0  | 3    | 2    |
| ## Hornet Sportabout | 18.7 | 8   | 360.0 | 175 | 3.15 | 3.440 | 17.02 | 0  | 0  | 3    | 2    |

# References

- Practical Data Science with R, by Nina Zumel and John Mount
- R을 이용한 데이터 분석 실무, 서민구, 길벗