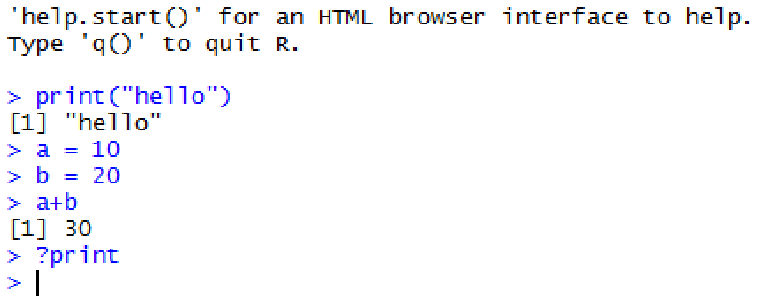
# R Basics Part 1

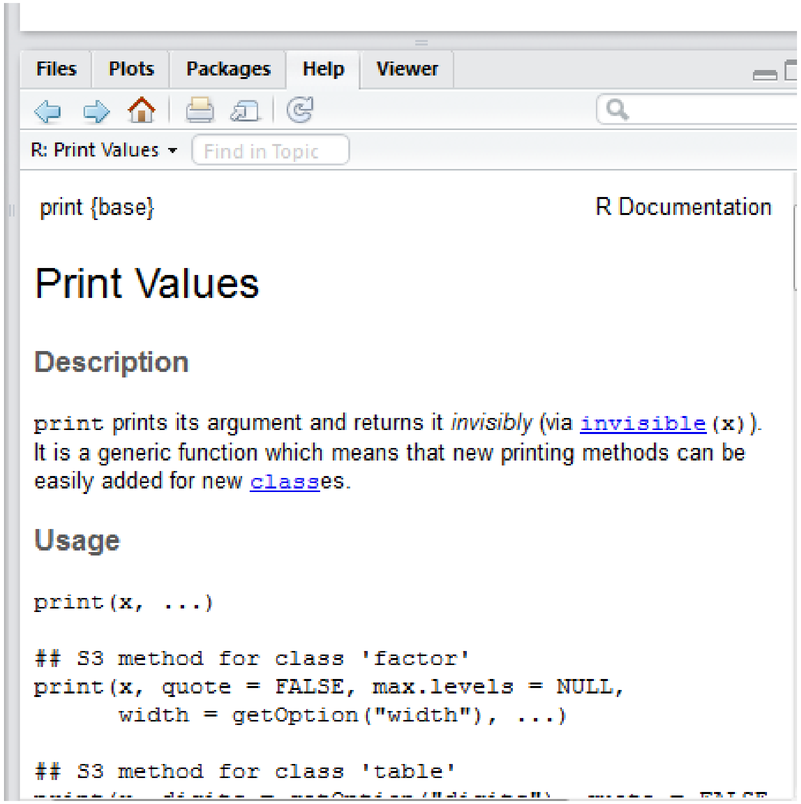
Presented by Hyebong Choi

# R

- Getting help
  - help(*command*) or ?*command*
  - example(*command*) to see examples



```
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> print("hello")
[1] "hello"
> a = 10
> b = 20
> a+b
[1] 30
> ?print
> |
```



print {base} R Documentation

## Print Values

### Description

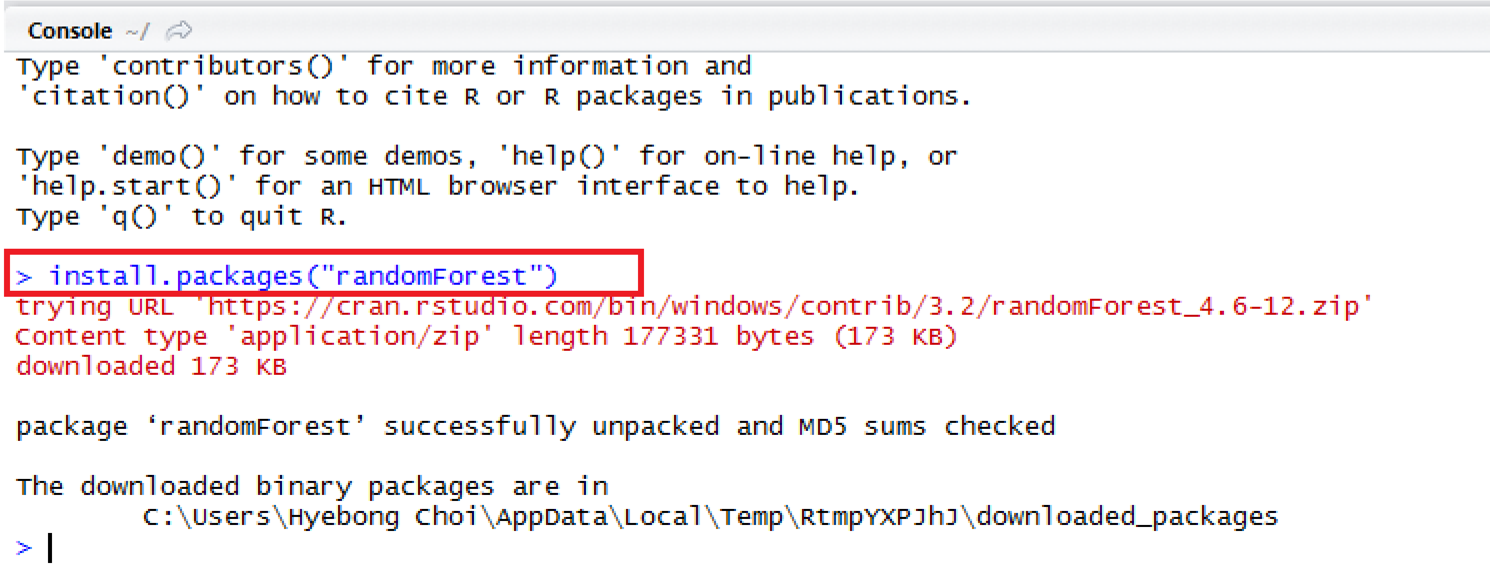
`print` prints its argument and returns it *invisibly* (via `invisible(x)`). It is a generic function which means that new printing methods can be easily added for new `class`es.

### Usage

```
print(x, ...)

## S3 method for class 'factor'
print(x, quote = FALSE, max.levels = NULL,
      width = getOption("width"), ...)

## S3 method for class 'table'
print(x, digits = getOption("digits"), quote = FALSE,
```

# Package Installation and loading

- install.packages(“*package name* ”)



```
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> install.packages("randomForest")
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.2/randomForest_4.6-12.zip'
Content type 'application/zip' length 177331 bytes (173 KB)
downloaded 173 KB

package ‘randomForest’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Hyebong Choi\AppData\Local\Temp\RtmpYXPJhJ\downloaded_packages
> |
```

- to load package
  - library(“*package name* ”)
  - or require(“*package name* ”)

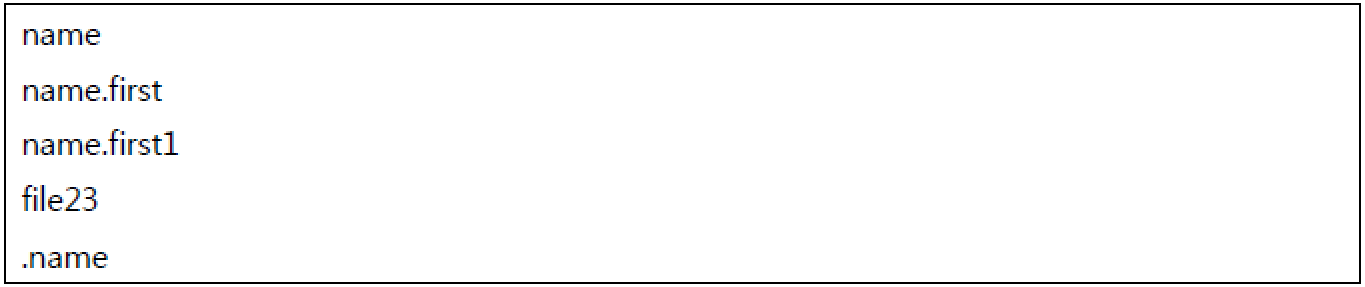
# Variable

- Variable is a container to hold data (or information) that we want to work with
- Variable can hold
  - a single value: 10, 10.5, “abc”, factor, NA, NULL
  - multiple values: vector, matrix, list
  - specially formatted data (values): data.frame

# Variable name should be …

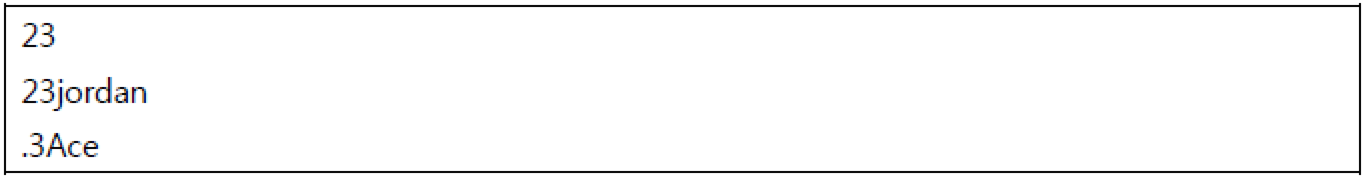
Naming rule
1. Consist of alphabet letter, '.' (dot), '_' (underscore) only.
2. First letter should be alphabet letter or dot('.')
3. Second letter after '.' cannot be numeric letter.

valid variable names



```
name
name.first
name.first1
file23
.name
```

invalid variable names

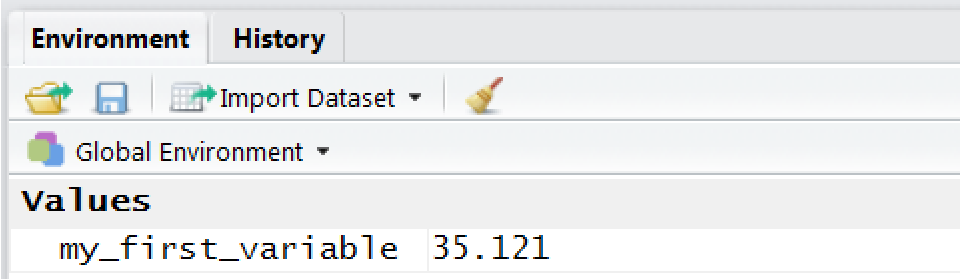


```
23
23jordan
.3Ace
```

# How you assign a value to variable

- var <- value

```
my_first_variable <- 35.121
```



```
Values
  my_first_variable  35.121
```

New variable is now assigned and available in working environment

# Data Type – Scala (Atomic Data Type)

R works with numerous data types. Some of the most basic types are:

- Decimals values like 4.5 are called numerics.
- Natural numbers like 4 are called integers. Integers are also numerics.
- Boolean values (TRUE or FALSE) are called logical.
- Text (or string) values are called characters.

Logical

```
typeof(TRUE)

## [1] "logical“
```

Double (i.e. numeric w/ decimal)

```
typeof(3.14)

## [1] "double"
```

Character (i.e. string)

```
typeof("hello")

## [1] "character"
```

Integer (i.e. numeric w/o decimal)

```
typeof(1L)

## [1] "integer"
```

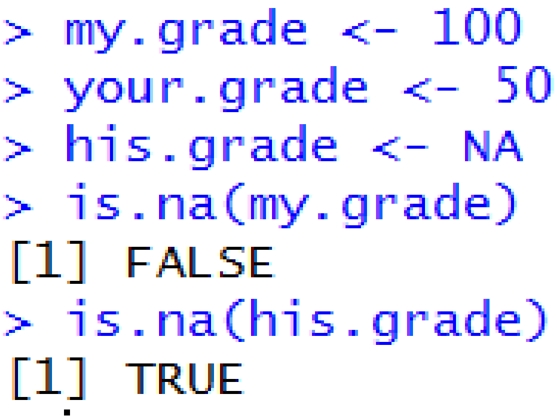
# Operators

```
a <- 10.5
b <- 20
c <- 4

a + b   ## addition

## [1] 30.5

a - c   ## substraction

## [1] 6.5

a * c   ## mulitiplication

## [1] 42

b / c   ## division

## [1] 5

a %% c  ## remainder

## [1] 2.5
```

```
a > b     ## inequality

## [1] FALSE

a*2 == b  ## equality

## [1] FALSE

!(a > b)  ## negation

## [1] TRUE

(b > a) & (b > c)  ## logical AND

## [1] TRUE

(a > b) | (a > c)  ## logical OR

## [1] TRUE
```

# Data Type – Missing Value (NA)

- Sometimes values are missing, and R represent the missing values as NAs



```
> my.grade <- 100
> your.grade <- 50
> his.grade <- NA
> is.na(my.grade)
[1] FALSE
> is.na(his.grade)
[1] TRUE
```

# Data type of some special values

```
typeof(Inf)                Infinity
## [1] "double"
typeof(-Inf)               Minus Infinity
## [1] "double"
typeof(NA)                 Missing Value
## [1] "logical"
```

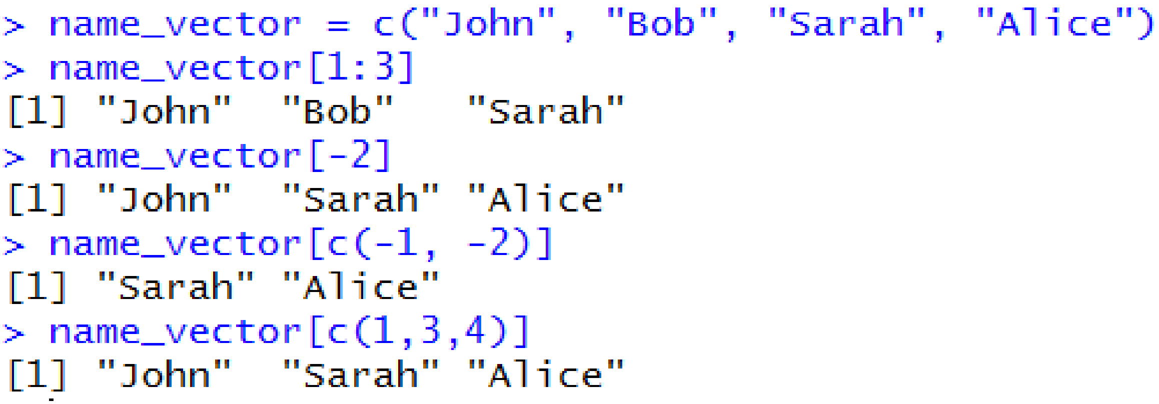
# Vector

- A **vector** is a sequence of data elements of the same basic type.
- All members should be of same data type

```
numeric_vector <- c(1, 10, 49)
character_vector <- c("a", "b", "c")
boolean_vector <- c(TRUE, FALSE, TRUE)

typeof(numeric_vector)

## [1] "double"

typeof(character_vector)

## [1] "character"

typeof(boolean_vector)

## [1] "logical"

length(numeric_vector)  ## number of members in the vector

## [1] 3

new_vector <- c(numeric_vector, 50)
new_vector

## [1]  1 10 49 50
```

# Vector

- R’s vector index starts from 1
  - 1,2,3,4, …
- Minus Index means “except for”



```
> name_vector = c("John", "Bob", "Sarah", "Alice")
> name_vector[1:3]
[1] "John"  "Bob"   "Sarah"
> name_vector[-2]
[1] "John"  "Sarah" "Alice"
> name_vector[c(-1, -2)]
[1] "Sarah" "Alice"
> name_vector[c(1,3,4)]
[1] "John"  "Sarah" "Alice"
```

# Vector with named elements

- We can give name to each element of vector
- and we can use the name instead of index number

```
some_vector <- c("John Doe", "poker player")
names(some_vector) <- c("Name", "Profession")

some_vector

##            Name     Profession
##      "John Doe" "poker player"
```

```
some_vector['Name']

##       Name
## "John Doe"

some_vector['Profession']

##     Profession
## "poker player"

some_vector[1]

##       Name
## "John Doe"
```

# Vector with named elements

- We can give name to each element of vector
- and we can use the name instead of index number

```
weather_vector <- c("Mon" = "Sunny", "Tues" = "Rainy",
                    "Wed" = "Cloudy", "Thur" = "Foggy",
                    "Fri" = "Sunny", "Sat" = "Sunny",
                    "Sun" = "Cloudy")
```

```
weather_vector
```

```
##      Mon     Tues      Wed     Thur      Fri      Sat      Sun
##  "Sunny"  "Rainy" "Cloudy"  "Foggy"  "Sunny"  "Sunny" "Cloudy“
```

```
names(weather_vector)
```

```
## [1] "Mon"  "Tues" "Wed"  "Thur" "Fri"  "Sat"  "Sun"
```

# Short-cut to make numeric vector

```
a_vector <- 1:10          ## numbers from 1 to 10
b_vector <- seq(1, 10, 2) ## numbers from 1 to 10 increasing by 2

a_vector

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

b_vector

## [1] 1 3 5 7 9

c_vector <- rep(1:3, 3)
d_vector <- rep(1:3, each = 3)

c_vector

## [1] 1 2 3 1 2 3 1 2 3

d_vector

## [1] 1 1 1 2 2 2 3 3 3

c(a_vector, b_vector) ## combine vectors to single vector

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

# Question

- What happen when you combine two vectors with different data type?

# Some useful functions

```
a_vector <- c(1,5,2,7,8, 2, 3)
b_vector <- seq(1, 10, 3)

intersect(a_vector, b_vector) ## intersection

## [1] 1 7

union(a_vector, b_vector)     ## union

## [1]  1  5  2  7  8  3  4 10

setdiff(a_vector, b_vector)   ## set difference

## [1] 5 2 8 3

unique(a_vector)  ## find distinct members

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

# Basic Vector operations

```
a_vector <- c(1,5,2,7,8)
b_vector <- seq(1, 10, 2)

sum(a_vector)  ## summation

## [1] 23

mean(a_vector) ## average

## [1] 4.6
```

```
# operation of Vector and Scala
a_vector + 10

## [1] 11 15 12 17 18

a_vector > 4

## [1] FALSE  TRUE FALSE  TRUE  TRUE

sum(a_vector > 4) ## what does this mean?

## [1] 3
```

```
# operation of Vector and Vector
a_vector - b_vector

## [1]  0  2 -3  0 -1

a_vector == b_vector

## [1]  TRUE FALSE FALSE  TRUE FALSE

sum(a_vector == b_vector) ## what does this mean?

## [1] 2
```

# Question

- What happen when we perform operation on two vectors with different length?

# Vector Indexing (Selection)

```
sample_vector <- c(1, 4, NA, 2, 1, NA, 4, NA) ## vector with some missing values
```

Selection by numeric vector

```
sample_vector[1:5]
## [1]  1  4 NA  2  1
sample_vector[c(1,3,5)]
## [1]  1 NA  1
sample_vector[-1]
## [1]  4 NA  2  1 NA  4 NA
sample_vector[c(-1, -3, -5)]
## [1]  4  2 NA  4 NA
```

Selection by logical vector

```
sample_vector[c(T, T, F, T, F, T, F, T)]
## [1]  1  4  2 NA NA
is.na(sample_vector)
## [1] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE
sum(is.na(sample_vector))
## [1] 3
## can you select non-NA elements from the vector?
```

# Matrix

In R, a matrix is a collection of elements of the same data type (numeric, character, or logical) arranged into a fixed number of rows and columns. Since you are only working with rows and columns, a matrix is called two-dimensional.

You can construct a matrix in R with the matrix() function.

```
matrix(1:9, byrow = TRUE, nrow = 3)
```

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

# Matrix

```
matrix(1:9, byrow = TRUE, nrow = 3)
```

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

In the matrix() function:

The first argument is the collection of elements that R will arrange into the rows and columns of the matrix.

Here, we use 1:9 which is a shortcut for c(1, 2, 3, 4, 5, 6, 7, 8, 9).

The argument byrow indicates that the matrix is filled by the rows. If we want the matrix to be filled by the columns, we just place byrow = FALSE.

The third argument nrow indicates that the matrix should have three rows.

# Naming a matrix

- Similar to vectors, you can add names for the rows and the columns of a matrix

```
rownames(my_matrix) <- row_names_vector
colnames(my_matrix) <- col_names_vector
```

# Naming a matrix

```
# Box office Star Wars (in millions!)
new_hope <- c(460.998, 314.4)
empire_strikes <- c(290.475, 247.900)
return_jedi <- c(309.306, 165.8)

# Construct matrix
star_wars_matrix <- matrix(c(new_hope, empire_strikes, return_jedi), nrow = 3, byrow = TRUE)

star_wars_matrix

##         [,1]  [,2]
## [1,] 460.998 314.4
## [2,] 290.475 247.9
## [3,] 309.306 165.8

# Vectors region and titles, used for naming
region <- c("US", "non-US")
titles <- c("A New Hope", "The Empire Strikes Back", "Return of the Jedi")

# Name the columns with region
colnames(star_wars_matrix) <- region

# Name the rows with titles
rownames(star_wars_matrix) <- titles

star_wars_matrix

##                              US non-US
## A New Hope              460.998  314.4
## The Empire Strikes Back 290.475  247.9
## Return of the Jedi      309.306  165.8
```

# row-wise and column-wise summation

```
# The worldwide box office figures
rowSums(star_wars_matrix)
```

```
##               A New Hope The Empire Strikes Back      Return of the Jedi 
##                  775.398                 538.375                 475.106
```

```
# Total revenue for entire Series
colSums(star_wars_matrix)
```

```
##       US   non-US 
## 1060.779  728.100
```

# Adding new column with cbind

```
# The worldwide box office figures
worldwide_vector <- rowSums(star_wars_matrix)

# Bind the new variable worldwide_vector as a column to star_wars_matrix
all_wars_matrix <- cbind(star_wars_matrix, worldwide_vector)

all_wars_matrix

##                              US non-US worldwide_vector
## A New Hope              460.998  314.4          775.398
## The Empire Strikes Back 290.475  247.9          538.375
## Return of the Jedi      309.306  165.8          475.106
```

# Adding new rows with rbind

```
# Construct star_wars_matrix2
box_office <- c(474.5, 552.5, 310.7, 338.7, 380.3, 468.5)
star_wars_matrix2 <- matrix(box_office, nrow = 3, byrow = TRUE,
                           dimnames = list(c("The Phantom Menace", "Attack of the Clones", "Revenge of
the Sith"),
                                           c("US", "non-US")))

star_wars_matrix

##                               US non-US
## A New Hope                 460.998  314.4
## The Empire Strikes Back 290.475  247.9
## Return of the Jedi      309.306  165.8

star_wars_matrix2

##                          US non-US
## The Phantom Menace   474.5   552.5
## Attack of the Clones 310.7   338.7
## Revenge of the Sith  380.3   468.5

all_wars_matrix <- rbind(star_wars_matrix, star_wars_matrix2)

all_wars_matrix

##                               US non-US
## A New Hope              460.998  314.4
## The Empire Strikes Back 290.475  247.9
## Return of the Jedi      309.306  165.8
## The Phantom Menace      474.500  552.5
## Attack of the Clones    310.700  338.7
## Revenge of the Sith     380.300  468.5
```

# Selection of matrix elements

Similar to vectors, you can use the square brackets [ ] to select one or multiple elements from a matrix. Whereas vectors have one dimension, matrices have two dimensions. You should therefore use a comma to separate that what to select from the rows from that what you want to select from the columns. For example:

- my_matrix[1,2] selects the element at the first row and second column.
- my_matrix[1:3,2:4] results in a matrix with the data on the rows 1, 2, 3 and columns 2, 3, 4.

If you want to select all elements of a row or a column, no number is needed before or after the comma, respectively:

- my_matrix[,1] selects all elements of the first column.
- my_matrix[1,] selects all elements of the first row.

# Selection of matrix elements

```
all_wars_matrix
```

```
##                              US non-US
## A New Hope              460.998  314.4
## The Empire Strikes Back 290.475  247.9
## Return of the Jedi      309.306  165.8
## The Phantom Menace      474.500  552.5
## Attack of the Clones    310.700  338.7
## Revenge of the Sith     380.300  468.5
```

```
all_wars_matrix[1:3,1]
```

```
##              A New Hope The Empire Strikes Back      Return of the Jedi
##                 460.998                 290.475                 309.306
```

```
all_wars_matrix[1:3,'non-US']
```

```
##              A New Hope The Empire Strikes Back      Return of the Jedi
##                   314.4                   247.9                   165.8
```

```
all_wars_matrix[,'US']
```

```
##              A New Hope The Empire Strikes Back      Return of the Jedi
##                 460.998                 290.475                 309.306
##      The Phantom Menace    Attack of the Clones     Revenge of the Sith
##                 474.500                 310.700                 380.300
```

```
all_wars_matrix[c(1,3,5),]
```

```
##                           US non-US
## A New Hope           460.998  314.4
## Return of the Jedi   309.306  165.8
## Attack of the Clones 310.700  338.7
```

# Some computation on matrices

```
A.mat <- matrix(1:9, byrow = TRUE, nrow = 3)
B.mat <- matrix(rep(1:3,each = 3), byrow = TRUE, nrow = 3)
C.mat <- matrix(rep(1:3, 2), byrow = F, ncol = 2)
```

```
A.mat
```

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

```
B.mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    2    2    2
## [3,]    3    3    3
```

```
C.mat
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    2
## [3,]    3    3
```

```
## matrix operation with scala
A.mat * 2
```

```
##      [,1] [,2] [,3]
## [1,]    2    4    6
## [2,]    8   10   12
## [3,]   14   16   18
```

```
A.mat - 10
```

```
##      [,1] [,2] [,3]
## [1,]   -9   -8   -7
## [2,]   -6   -5   -4
## [3,]   -3   -2   -1
```

```
A.mat / 5
```

```
##      [,1] [,2] [,3]
## [1,]  0.2  0.4  0.6
## [2,]  0.8  1.0  1.2
## [3,]  1.4  1.6  1.8
```

# Some computation on matrices

```
A.mat
```

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

```
B.mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    2    2    2
## [3,]    3    3    3
```

```
C.mat
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    2
## [3,]    3    3
```

```
## matrix operation with other matrix
## (element-wise operation)
A.mat * B.mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    8   10   12
## [3,]   21   24   27
```

```
A.mat - B.mat
```

```
##      [,1] [,2] [,3]
## [1,]    0    1    2
## [2,]    2    3    4
## [3,]    4    5    6
```

```
A.mat / B.mat
```

```
##          [,1]     [,2] [,3]
## [1,] 1.000000 2.000000    3
## [2,] 2.000000 2.500000    3
## [3,] 2.333333 2.666667    3
```

# Some computation on matrices

```
A.mat
```

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

```
B.mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    2    2    2
## [3,]    3    3    3
```

```
C.mat
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    2
## [3,]    3    3
```

```
## matrix multiplication
A.mat %*% C.mat
```

```
##      [,1] [,2]
## [1,]   14   14
## [2,]   32   32
## [3,]   50   50
```

# Factor

- R represents categorical variables as factor
  - e.g. gender in one of (male, female)
  - other examples, major, nationality, blood type, etc.
  - belong to a limited number of categories

```r
# Sex vector
sex_vector <- c("Male", "Female", "Female", "Male", "Male")

# Convert sex_vector to a factor
factor_sex_vector <- factor(sex_vector)

# Print out factor_sex_vector
print(factor_sex_vector)
```

```
## [1] Male   Female Female Male   Male
## Levels: Female Male
```

# Factor

Factor holds integers as value with level information

```
typeof(factor_sex_vector)

## [1] "integer"

str(factor_sex_vector)

##  Factor w/ 2 levels "Female","Male":
2 1 1 2 2

levels(factor_sex_vector)

## [1] "Female" "Male"
```

# Changing Levels

Recording the sex with the abbreviations "M" and "F" can be convenient if you are collecting data with pen and paper, but it can introduce confusion when analyzing the data. At that point, you will often want to change the factor levels to "Male" and "Female" instead of "M" and "F" for clarity.

```
# Code to build factor_survey_vector
survey_vector <- c("M", "F", "F", "M", "M")
factor_survey_vector <- factor(survey_vector)
factor_survey_vector
```

```
## [1] M F F M M
## Levels: F M
```

```
# Specify the levels of factor_survey_vector
levels(factor_survey_vector) <- c('Female', 'Male')

factor_survey_vector
```

```
## [1] Male   Female Female Male   Male
## Levels: Female Male
```

# Summarizing a Factor

summary() function gives you a quick overview of the contents of a variable

```
# Generate summary for survey_vector
summary(survey_vector)
```

```
##    Length     Class      Mode
##         5 character character
```

```
# Generate summary for factor_survey_vector
summary(factor_survey_vector)
```

```
## Female   Male
##      2      3
```

# Data Frame

- Very commonly datasets contains variables of different kinds
  - e.g. student dataset may contain name(character), age(integer), major(factor), gpa(numeric, real number)…
- Vector and metric can have values of same data type
- A data frame has the variables of a data set as columns and the observations as rows.

```
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
```

# Overviewing of Data frame

- head functions shows the first n (6 by default) observation of dataframe
- tail functions shows the last n (6 by default) observation of dataframe

```
head(mtcars)

##                    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

head(mtcars, 10)  ## try to see what happens

tail(mtcars)

##                 mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2

tail(mtcars, 10) ## try to see what happens
```

# Overviewing of Data frame

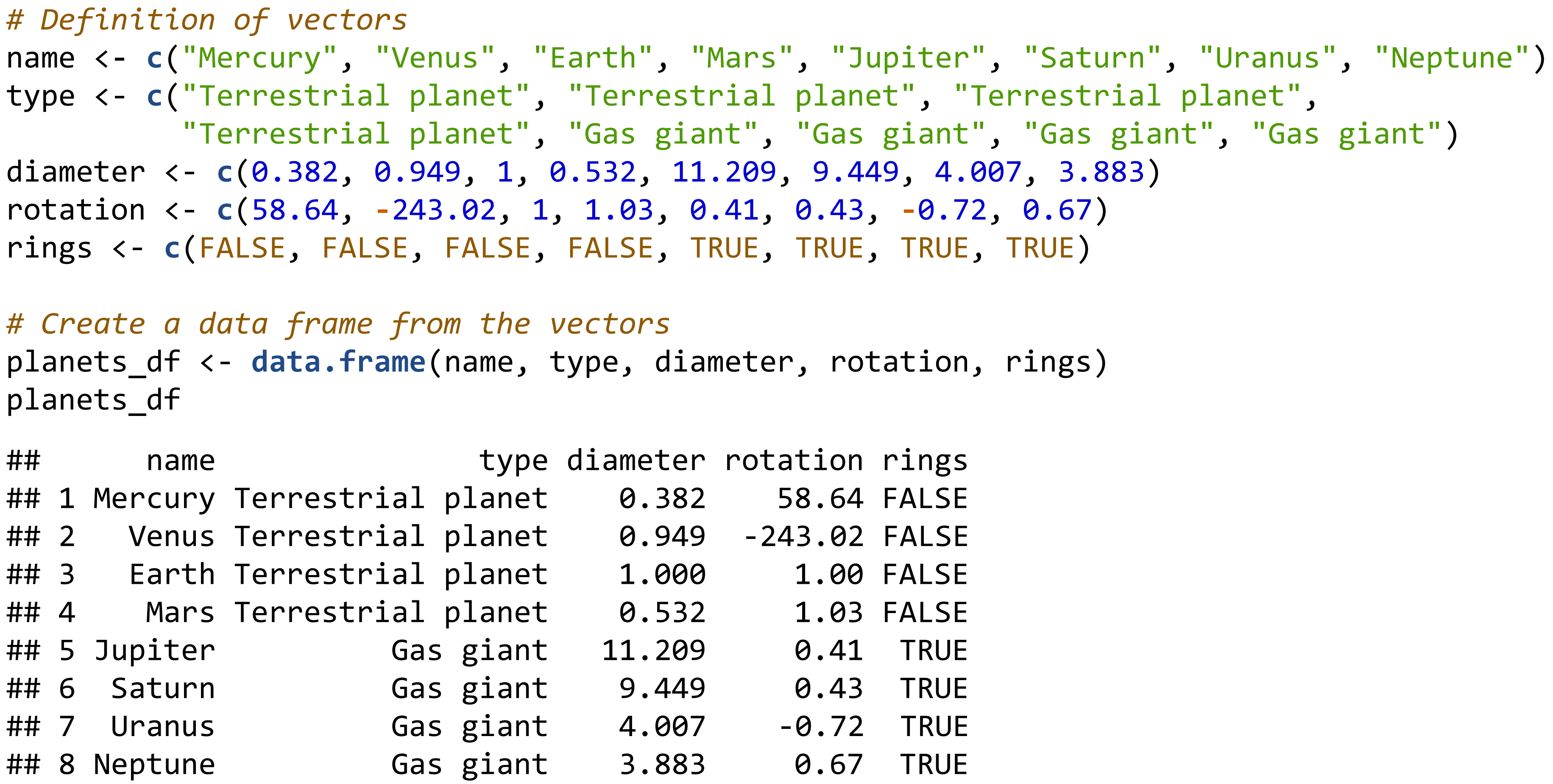
- str function shows the structure of your data set, it tells you
  - The total number of observations (e.g. 32 car types)
  - The total number of variables (e.g. 11 car features)
  - A full list of the variables names (e.g. mpg, cyl ... )
  - The data type of each variable (e.g. num)
  - The first few observations

```
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

# Creating Data frame

- data.frame function with vectors (of same length and possibly different type) makes you a data frame



```
# Definition of vectors
name <- c("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune")
type <- c("Terrestrial planet", "Terrestrial planet", "Terrestrial planet",
          "Terrestrial planet", "Gas giant", "Gas giant", "Gas giant", "Gas giant")
diameter <- c(0.382, 0.949, 1, 0.532, 11.209, 9.449, 4.007, 3.883)
rotation <- c(58.64, -243.02, 1, 1.03, 0.41, 0.43, -0.72, 0.67)
rings <- c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE)

# Create a data frame from the vectors
planets_df <- data.frame(name, type, diameter, rotation, rings)
planets_df
```

```
##      name               type diameter rotation rings
## 1 Mercury Terrestrial planet    0.382    58.64 FALSE
## 2   Venus Terrestrial planet    0.949  -243.02 FALSE
## 3   Earth Terrestrial planet    1.000     1.00 FALSE
## 4    Mars Terrestrial planet    0.532     1.03 FALSE
## 5 Jupiter          Gas giant   11.209     0.41  TRUE
## 6  Saturn          Gas giant    9.449     0.43  TRUE
## 7  Uranus          Gas giant    4.007    -0.72  TRUE
## 8 Neptune          Gas giant    3.883     0.67  TRUE
```

# Creating Data frame

- you may specify the variables as parameters

```
my.df <- data.frame(name = c('John', 'Kim', 'Kaith'), job =
c('Teacher', 'Policeman', 'Secertary'), age = c(32, 25, 28))
my.df
```

```
##    name       job age
## 1  John   Teacher  32
## 2   Kim Policeman  25
## 3 Keith Secretary  28
```

# Selection of data frame elements

Similar to vectors and matrices, you select elements from a data frame with the help of square brackets [ ].

By using a comma, you can indicate what to select from the rows and the columns respectively.

```
# Print out diameter of Mercury (row 1, column 3)
planets_df[1,3]
```

```
## [1] 0.382
```

```
# Print out data for Mars (entire fourth row)
planets_df[4, ]
```

```
##   name               type diameter rotation rings
## 4 Mars Terrestrial planet    0.532     1.03 FALSE
```

```
# you can use of directly variable name
# Select first 5 values of diameter column
planets_df[1:5, 'diameter']
```

```
## [1]  0.382  0.949  1.000  0.532 11.209
```

# Selection of data frame elements

You will often want to select an entire column, namely one specific variable from a data frame. If you want to select all elements of the variable diameter, for example, both of these will do the trick:

```
planets_df[,3]
planets_df[,"diameter"]
```

However, there is a short-cut. If your columns have names, you can use the $ sign:

```
planets_df$diameter
```

# Selection of data frame elements - a tricky part

- You can use a logical vector to select from data frame

```
## find planets with rings
planets_df[planets_df$rings, ]

##      name      type diameter rotation rings
## 5 Jupiter Gas giant   11.209     0.41  TRUE
## 6  Saturn Gas giant    9.449     0.43  TRUE
## 7  Uranus Gas giant    4.007    -0.72  TRUE
## 8 Neptune Gas giant    3.883     0.67  TRUE

## select names of planets with rings
planets_df[planets_df$rings, 'name']

## [1] Jupiter Saturn  Uranus  Neptune
## Levels: Earth Jupiter Mars Mercury Neptune Saturn Uranus Venus

## find planets with larger diameter than earth
planets_df$diameter > 1

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

planets_df[planets_df$diameter > 1, ]

##      name      type diameter rotation rings
## 5 Jupiter Gas giant   11.209     0.41  TRUE
## 6  Saturn Gas giant    9.449     0.43  TRUE
## 7  Uranus Gas giant    4.007    -0.72  TRUE
## 8 Neptune Gas giant    3.883     0.67  TRUE
```

# List

- A list in R allows you to gather a variety of objects under one name (that is, the name of the list) in an ordered way.
- These objects can be matrices, vectors, data frames, even other lists, etc.
- It is not even required that these objects are related to each other in any way.
  - Data frame can have variables(vectors) of same length (and possibly different types)
  - For list there is no such restriction
- To create a list, use list() function

```
my_list <- list(comp1, comp2 ...)
```

```
# Vector with numerics from 1 up to 10
my_vector <- 1:10

# Matrix with numerics from 1 up to 9
my_matrix <- matrix(1:9, ncol = 3)

# First 10 elements of the built-in data frame mtcars
my_df <- mtcars[1:10,]

# Construct list with these different elements:
my_list <- list(my_vector, my_matrix, my_df)

my_list

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

# List

- You can give names for each component in a list, so we may remember what components of list stand for

```
my_list <- list(name1 = your_comp1,
                name2 = your_comp2)
```

- We may change the name of each components with names() function

```
my_list <- list(your_comp1, your_comp2)
names(my_list) <- c("name1", "name2")
```

# List

```
# First 10 elements of the built-in data frame mtcars
my_df <- mtcars[1:3,]

# Adapt list() call to give the components names
my_list <- list(vec = my_vector, mat = my_matrix, df = my_df)

# Print out my_list
my_list
```

```
## $vec
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $mat
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## $df
##                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
```

# Selecting elements from a list

- One way to select a component is using the numbered position of that component with double square brakets [[ ]]
- You can also refer to the names of the components, with [[ ]] or with the $ sign.

```
my_list[[1]]
```

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

```
my_list[['mat']]
```

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

```
my_list$mat
```

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

# Adding more components to the list

```
my_list$new_vector <- c(1,3,5,7,9)

str(my_list)
```

```
## List of 4
##  $ vec       : int [1:10] 1 2 3 4 5 6 7 8 9 10
##  $ mat       : int [1:3, 1:3] 1 2 3 4 5 6 7 8 9
##  $ df        :'data.frame':  3 obs. of  11 variables:
##   ..$ mpg : num [1:3] 21 21 22.8
##   ..$ cyl : num [1:3] 6 6 4
##   ..$ disp: num [1:3] 160 160 108
##   ..$ hp  : num [1:3] 110 110 93
##   ..$ drat: num [1:3] 3.9 3.9 3.85
##   ..$ wt  : num [1:3] 2.62 2.88 2.32
##   ..$ qsec: num [1:3] 16.5 17 18.6
##   ..$ vs  : num [1:3] 0 0 1
##   ..$ am  : num [1:3] 1 1 1
##   ..$ gear: num [1:3] 4 4 4
##   ..$ carb: num [1:3] 4 4 1
##  $ new_vector: num [1:5] 1 3 5 7 9
```

```
my_list[['new_vector']]
```

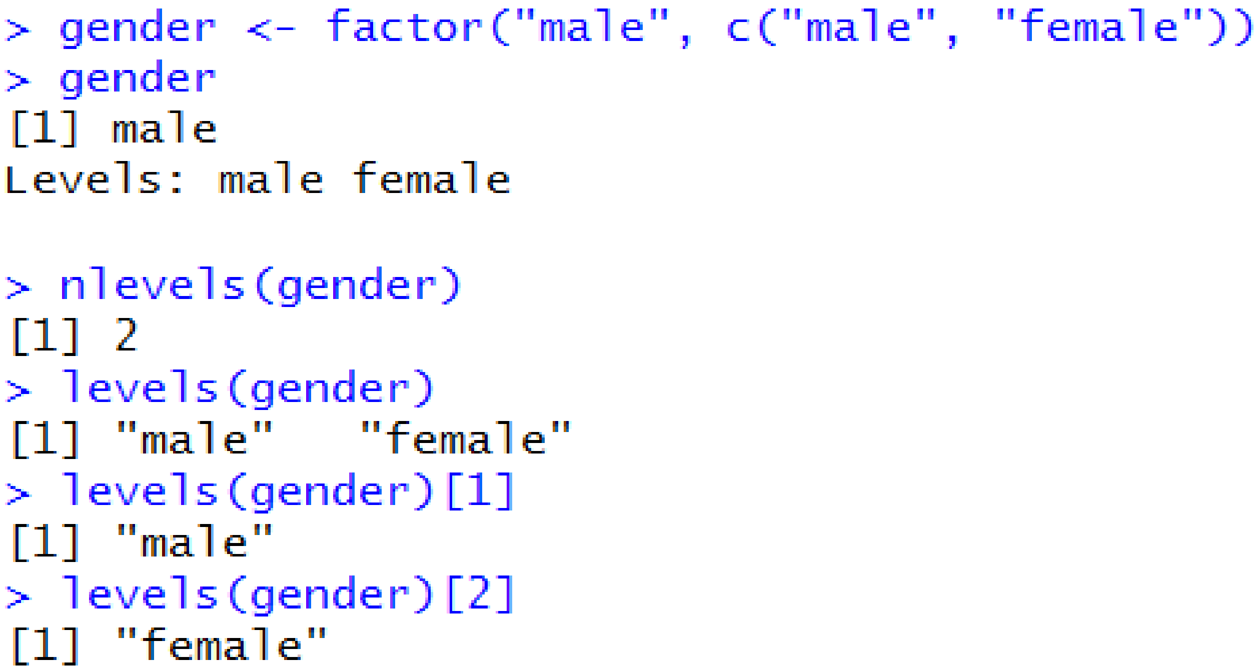
```
## [1] 1 3 5 7 9
```

# References

- Practical Data Science with R, by Nina Zumel and John Mount

# Data Type - Scala

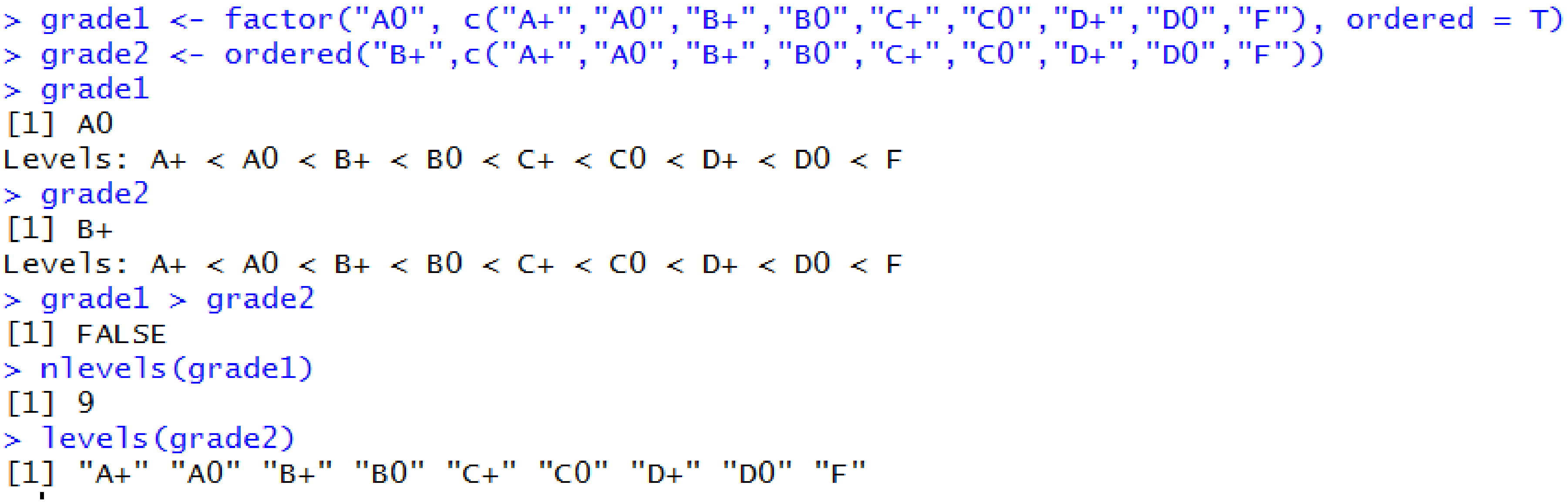
- Factor
  - Categorical data



```
> gender <- factor("male", c("male", "female"))
> gender
[1] male
Levels: male female

> nlevels(gender)
[1] 2
> levels(gender)
[1] "male"    "female"
> levels(gender)[1]
[1] "male"
> levels(gender)[2]
[1] "female"
```

# Data Type - Scala

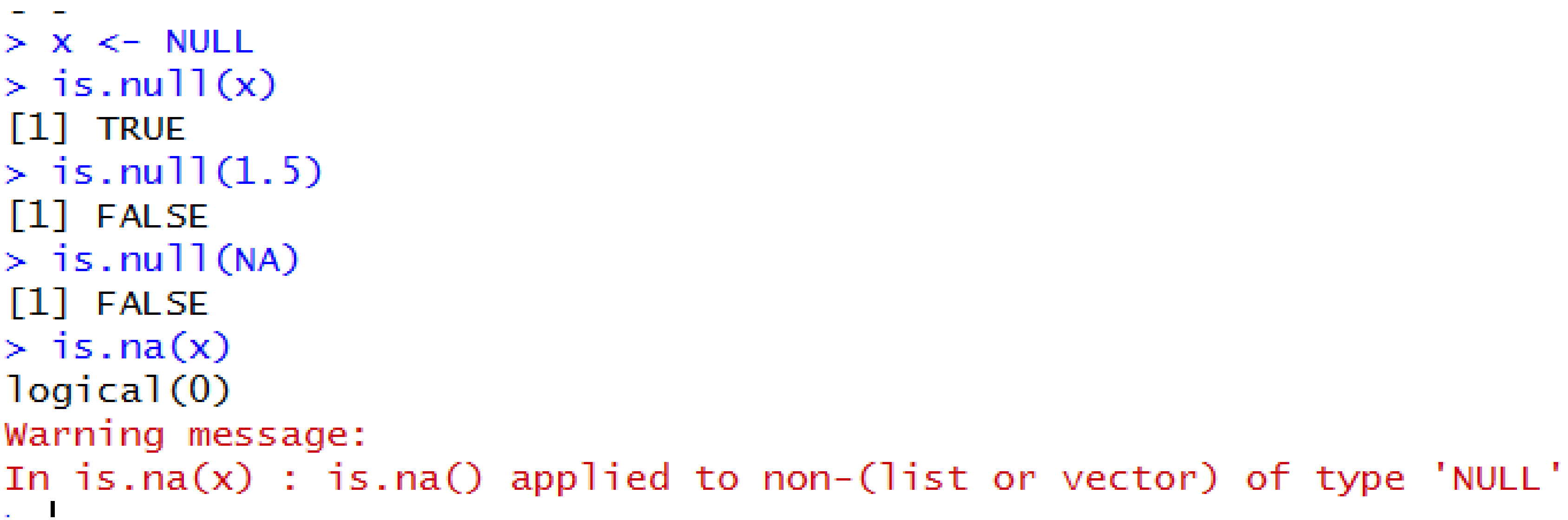
- Ordered Factor



```
> grade1 <- factor("A0", c("A+","A0","B+","B0","C+","C0","D+","D0","F"), ordered = T)
> grade2 <- ordered("B+",c("A+","A0","B+","B0","C+","C0","D+","D0","F"))
> grade1
[1] A0
Levels: A+ < A0 < B+ < B0 < C+ < C0 < D+ < D0 < F
> grade2
[1] B+
Levels: A+ < A0 < B+ < B0 < C+ < C0 < D+ < D0 < F
> grade1 > grade2
[1] FALSE
> nlevels(grade1)
[1] 9
> levels(grade2)
[1] "A+" "A0" "B+" "B0" "C+" "C0" "D+" "D0" "F"
```

# Data Type – NULL

- different with NA
  - NA is a value of missing
  - NULL means “not defined”, “not initialized”, or “not ready for use”



```
> x <- NULL
> is.null(x)
[1] TRUE
> is.null(1.5)
[1] FALSE
> is.null(NA)
[1] FALSE
> is.na(x)
logical(0)
Warning message:
In is.na(x) : is.na() applied to non-(list or vector) of type 'NULL'
```