Preview

- Let's learn about the grammar of data science.
  - **Variable**: data storage space
  - **Data types**: numeric, character, categorical, logical, special constants, etc.
  - **Operators**: arithmetic, comparison, logical operators
  - **Vector**: a collection of single values
  - **Array**: A set of data with columns and rows (or A set of vectors)
  - **Data frame**: A structure in which different data types are organized in a tabular form. Each property has the same size.
  - **List**: A tabular structure similar to "Data frame". The size of each property can be different.

01 Data storage and processing

- Grammar study is essential to save data and process operations
  - a=1
  - b=2
  - c=a+b

- When there needs a lot of data, such as student grade processing
  - A single variable cannot represent all the data
  - By using *vector*, *matrix*, *data frame*, *list*, etc., it is possible to store a lot of data with one variable name.
  - There are many things around us are organized in a tabular form for easy data management. (e.g. attendance checking, grade, and member management, etc.)

02 Variable

- Storing values in variables
  - Value assignment using =, <-, ->

```
> x = 1            # Assign 1 to X
> y = 2            # Assign 2 to Y.
> z = x + y
> z
[1] 3
> x + y = z
Error in x+y=z : could not find function "+<-"
> z <- x + y
> z
[1] 3
> x + y -> z
> z
[1] 3
```

02 Variable

Example of exchanging two values

- Make temporary storage space and save one value in advance
- The programs are executed sequentially from top to bottom.

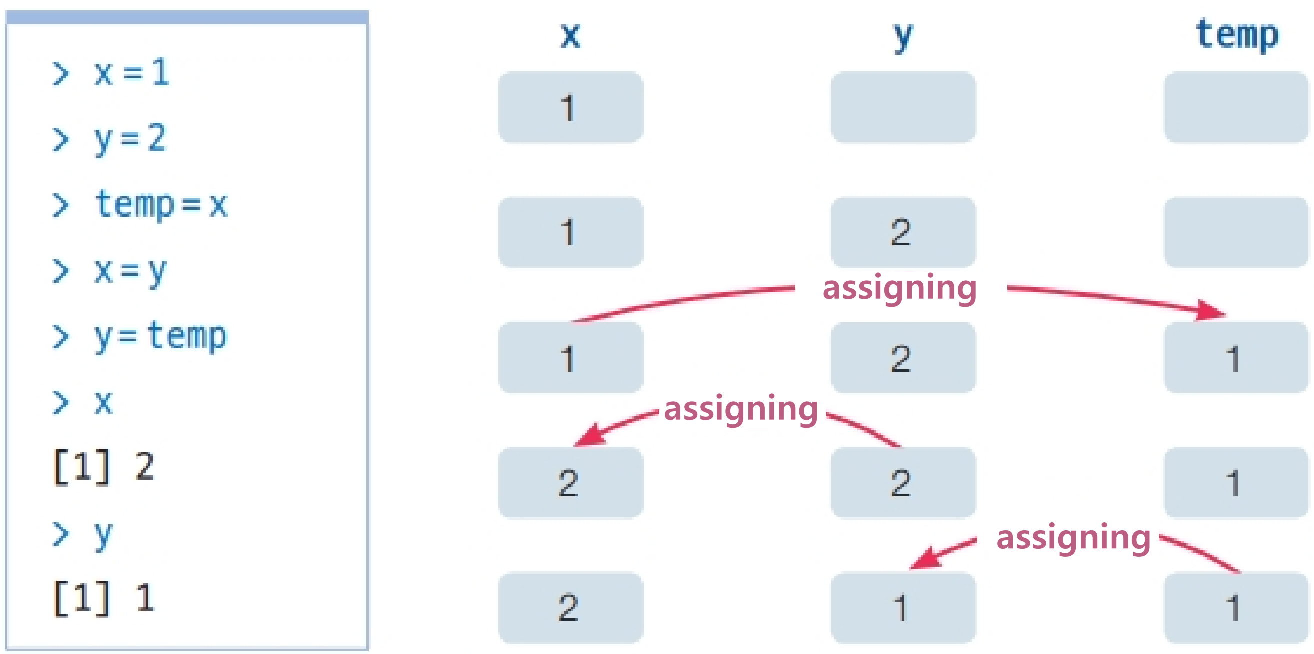


Figure 3-1 Source code for the exchange of two values, operating process

03 Data type

- Basic data types of R

Table 3-1 Basic Data Types Frequently Used by R

Data type	Kinds
Numeric	intinteger / numnumeric / cplxcomplex
Character	chrcharcter : String should be enclosed in single quotes or double quotes
Categorical	factor : Forms classified by level.
Logical	TRUE(T), FALSE(F)
Special constant	NULL : undefined value NANot Available : missing value -Inf (Negative infinity) and Inf (Positive infinity) NaNNot a Number : Values that cannot be computed, such as 0/0, Inf/Inf, etc

03 Data type

- Learning examples for basic data types in R

```
> x = 5
> y = 2
> x/y
[1] 2.5

> xi = 1 + 2i
> yi = 1 - 2i
> xi+yi
[1] 2+0i


> str = "Hello, World!"
> str
[1] "Hello, World!"
```

```
> blood.type = factor(c('A', 'B', 'O', 'AB'))
> blood.type
[1] A  B  O  AB
Levels: A AB B O

> T
[1] TRUE
> F
[1] FALSE

> xinf = Inf
> yinf = -Inf
> xinf/yinf
[1] NaN
```

03 Data type

Data type verification and conversion functions

Table 3-2 Functions to check data type

Function	Explanation
class(x)	Data type of X from an object-oriented perspective in R
typeof(x)	Data type of X from the R language's own perspective
is.integer(x)	True if X is an integer number type, False if it's not
is.numeric(x)	True if X is a real number type, False if it's not
is.complex(x)	True if X is a complex number type, False if it's not
is.character(x)	True if X is a character type, False if it's not
is.na(x)	True if X is NA type, False if it's not

Table 3-3 Functions to transform data type

Function	Explanation
as.factor(x)	Transform X to categorical type.
as.integer(x)	Transform X to integer type.
as.numeric(x)	Transform X to numerical type.
as.character(x)	Transform X to character type.
as.matrix(x)	Transform X to matrix.
as.array(x)	Transform X to array.

04 Operators

Type of operators

- Arithmetic operators, comparison operators, logical operators

Table 3-4 Arithmetic operators

Operator	Explanation	Example
+	Addition	5 + 2 → 7
−	Subtraction	5 − 2 → 3
*	Multiplication	5 * 2 → 10
/	Division(real number)	5 / 2 → 2.5
^ or **	exponent	5 ^ 2 → 25
x %% y	Remainder that X divided by Y (remainder of integer division)	5 %% 2 → 1
x %/% y	Quotient that X divided by Y (quotient of integer division)	5 %/% 2 → 2

Table 3-5 Comparison operators and Logical operators

Operator	Explanation	Example
<	left is less than right.	5 < 5 → FALSE
<=	left is less than right or the same.	5 <= 5 → TRUE
>	left is greater than right.	5 > 5 → FALSE
>=	left is greater than right or the same.	5 >= 5 → TRUE
==	left equal to right.	5 == 5 → TRUE
!=	left not equal to right .	5 != 5 → FALSE
!x	not	!TRUE → FALSE
x | y, x || y	X or y(or union)	TRUE | FALSE → TRUE
x & y, x && y	X and y (and intersection)	TRUE & FALSE → FALSE
isTRUE(x)	validating whether x is true or not	isTRUE(TRUE) → TRUE

04 Operators

- Operator priority

Figure 3-6 Operator Priority

<table>
<tr><th>Operator</th><th>Explanation</th><th>Priority</th></tr>
<tr><td>^, **</td><td>exponent</td><td></td></tr>
<tr><td>+, −</td><td>unary plus and minus</td><td></td></tr>
<tr><td>%any%</td><td>operators such as %% and %/%</td><td>High</td></tr>
<tr><td>*, /</td><td>multiplication, division</td><td></td></tr>
<tr><td>+, -</td><td>addition, subtraction</td><td></td></tr>
<tr><td>==, !=, <, >, <=, >=</td><td>comparison operator</td><td></td></tr>
<tr><td>!</td><td>negative of logic (not)</td><td>Low</td></tr>
<tr><td>&, &&</td><td>logic and</td><td></td></tr>
<tr><td>|, ||</td><td>logic or</td><td></td></tr>
</table>

05 Vector

- Multiple single values can be stored as one variable name
  - Storing a single value as a single variable increases the number of variables if there are many values
  - Multiple single values can be stored as a single vector variable.

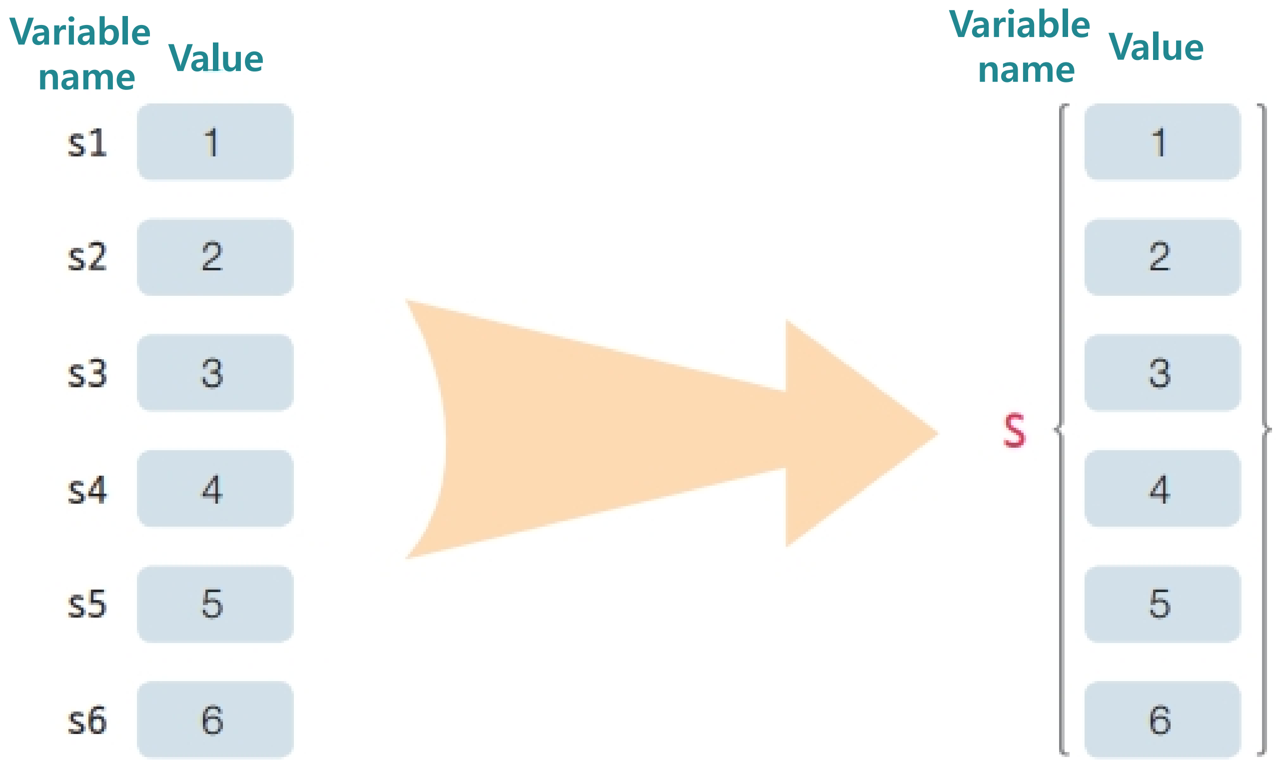


Figure 3-2 A vector consisting of a combination of single values

05 Vector

Vector generation

- Use vector generation operator ':'

```
> 1:7     # Increase by 1, from 1 to 7 to generate a vector with 7 elements.
[1] 1 2 3 4 5 6 7

> 7:1     # Decrease by 1, from 1 to 7 to generate a vector with 7 elements.
[1] 7 6 5 4 3 2 1
```

- Use vector function
  - Create empty vector with n elements

Number of Elements

```
> vector(length = 5)
[1] FALSE FALSE FALSE FALSE FALSE
```

05 Vector

- Using c function: generating a generic vector

```
> c(1:5)                    # Vector generating consisting of 1 to 5 elements. equal to (1:5)
[1] 1 2 3 4 5

> c(1, 2, 3, c(4:6))        # Vector generating consisting of elements 1 to 6 that combine
                                      elements 1 to 3 and elements 4 to 6
[1] 1 2 3 4 5 6

> x = c(1, 2, 3)            # Storing a vector consisting of 1 to 3 elements in x
> x                         # the output of x
[1] 1 2 3

> y = c()                   # Generating y with an empty vector
> y = c(y, c(1:3))          # Generating a vector by adding a c (1:3) vector
                                      to an existing y vector
> y                         # the output of y
[1] 1 2 3
```

05 Vector

- Using the seq function: generating permutation vectors

Initial value　　End value　　Increasing value

```
> seq(from=1, to=10, by=2)     # Vector generating increasing by 2 from 1 to 10
[1] 1 3 5 7 9

> seq(1, 10, by=2)             # Vector generating increasing by 2 from 1 to 10
[1] 1 3 5 7 9
```

```
> seq(0, 1, by=0.1)            # Vector generating increasing by 0.1 from 0 to 1
                                 with 11 elements
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

Number of Elements

```
> seq(0, 1, length.out=11)     # Vector generating from 0 to 1 with 11 elements
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

05 Vector

- Using the rep function: generating iterative vectors

Vector Iterative count of vector

```
> rep(c(1:3), times=2)     # Generating a vector that iterates vector (1, 2, 3) twice.
[1] 1 2 3 1 2 3
> rep(c(1:3), each=2)      # Generating a vector that iterates the individual
                             elements of vector (1, 2, 3) twice.
[1] 1 1 2 2 3 3
```

Iterative count of element

05 Vector

Vector operation

- Select and print the elements of a vector

```
> x = c(2, 4, 6, 8, 10)
> length(x)                # Getting the length (size) of x vector
[1] 5

> x[1]                     # Getting the value of element number 1 of x vector
[1] 2

                           # Element 1, 2, 3 of x vector are not available
> x[1, 2, 3]                 due to an error
Error in x[1, 2, 3] : incorrect number of dimensions

                           # To get elements 1, 2, 3 of x vectors,
> x[c(1, 2, 3)]              you must bind them by a vector
[1] 2 4 6

> x[-c(1, 2, 3)]           # Output of all elements except elements 1, 2, 3
[1]  8 10                    from x vectors.

> x[c(1:3)]                # Output of elements from 1 through 3 from x vector.
[1] 2 4 6
```

05 Vector

- Inter-vector operation: Operation is possible when the lengths of the vectors are the same or the number of elements is multiple relationship with the opposite one

```
> x=c(1, 2, 3, 4)
> y=c(5, 6, 7, 8)
> z=c(3, 4)
> w=c(5, 6, 7)
```

```
> x+2          # Adding 2 to each of the individual elements of the x vector
[1] 3 4 5 6

> x+y          # The size (length) of the x vector and y vector are the same,
               so add each element
[1]  6  8 10 12

> x+z          # If the x vector is twice (in integer) the size of the z-vector,
               cycle the elements of the small vector and add them
[1] 4 6 6 8

> x+w          # An error appears because the size of x and w is not twice (in integer)
[1]  6  8 10  9
Warning message:
In x+w : longer object length is not a multiple of shorter object length
```

05 Vector

Useful functions for vector operations

- all • any function: Review the condition of all or some elements in a vector

```
> x=1:10

> x>5           # Validating that each element of x vector is greater than 5 or not
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE

> all(x>5)      # Validating that all elements of x vector are greater than 5 or not
[1] FALSE

> any(x>5)  # Validating that some of x vector's elements are greater than 5 or not
[1] TRUE
```

05 Vector

- head・tail functions: Extract some elements from the front and back of the data (six basic extracts)

```
> x=1:10

> head(x)        # Extract 6 elements from the front in data
[1] 1 2 3 4 5 6

> tail(x)        # Extract 6 elements from the back in data
[1]  5  6  7  8  9 10

> head(x, 3)     # Extract 3 elements from the front in data
[1] 1 2 3

> tail(x, 3)     # Extract 3 elements from the back in data
[1]  8  9 10
```

05 Vector

- union・intersect・setdiff・setequal fucntions: Set operation between vectors

```
> x=c(1, 2, 3)
> y=c(3, 4, 5)
> z=c(3, 1, 2)

> union(x, y)          # Union
[1] 1 2 3 4 5

> intersect(x, y)      # Intersection
[1] 3

> setdiff(x, y)        # Difference of sets
[1] 1 2                  (excluding elements equal to y in x)

> setdiff(y, x)        # Difference of sets
[1] 4 5                  (excluding elements equal to x in y)

> setequal(x, y)       # validating that x and y have the same
[1] FALSE                    elements each other

> setequal(x, z)       # validating that x and z have the same
[1] TRUE                     elements each other
```

06 Array

- Array: Data consisting of columns and rows

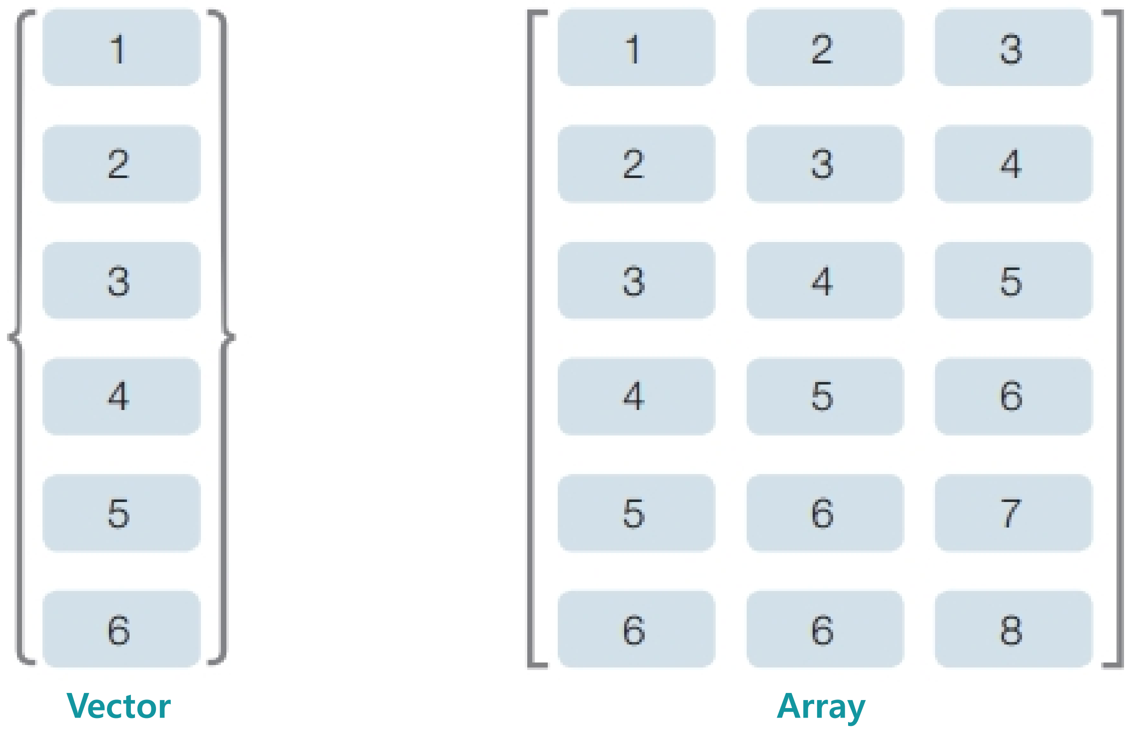


Figure 3-4 **Configurations of vectors and arrangements**

06 Array: Array generating functions

Array generating functions

- Array function : N-Dimensional array generating

Vector data

Vector defining dimensions

```
> # N-Dimensional Array Generating
> x = array(1:5, c(2, 4))     # Assign 1~5 values to 2x4 matrices
```

```
> x
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    2
[2,]    2    4    1    3

> x[1, ]    # Outputs an element value in a row
[1] 1 3 5 2

> x[, 2]    # Outputs the element values in column 2
[1] 3 4
```

06 Array: Array generating functions

- Matrix function : Generating 2-dimensional array

```
> # Generating 2-dimensional array
> x = 1:12
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

Vector to be configured as a matrix

Either the number of rows or the number of columns

```
> matrix(x, nrow = 3)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Whether to place data by row units (T/F)

```
> matrix(x, nrow = 3, byrow = T)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

Add a list of row and column names is availab le through 'dimname' option.

06 Array: Array generating functions

- cbind · rbind function : Generating an array by columns · rows

```
> # Generating an array by binding vector
> v1 = c(1, 2, 3, 4)
> v2 = c(5, 6, 7, 8)
> v3 = c(9, 10, 11, 12)

> cbind(v1, v2, v3)     # Generating an array by binding columns units
     v1 v2 v3
[1,]  1  5  9
[2,]  2  6 10
[3,]  3  7 11
[4,]  4  8 12

> rbind(v1, v2, v3)     # Generating an array by binding row units
   [,1] [,2] [,3] [,4]
v1    1    2    3    4
v2    5    6    7    8
v3    9   10   11   12
```

06 Array: Array computation

- Array(matrix) operator

Figure 3-7 Matrix operator

operator	Explanation
+, −	Addition and subtraction of a matrix
*	Matrix multiplication in R (for each column)
%*%	Mathematical matrix multiplication
t(), aperm()	Transposed matrix
solve()	Inverse matrix
det()	A determinant

06 Array: Array operation

- Array operation example

```
# Various matrix operations using operators of [Figure 3-7]

> # Storing two 2x2 matrices in x and y, respectively
> x = array(1:4, dim = c(2, 2))
> y = array(5:8, dim = c(2, 2))

> x
     [,1] [,2]
[1,]    1    3
[2,]    2    4

> y
     [,1] [,2]
[1,]    5    7
[2,]    6    8

> x + y
     [,1] [,2]
[1,]    6   10
[2,]    8   12

> x - y
     [,1] [,2]
[1,]   -4   -4
[2,]   -4   -4

> x * y                 # Multiplication for each column
     [,1] [,2]
[1,]    5   21
[2,]   12   32

> x %*% y               # Mathematical matrix multiplication
     [,1] [,2]
[1,]   23   31
[2,]   34   46

> t(x)                  # Transposed matrix of x
     [,1] [,2]
[1,]    1    2
[2,]    3    4

> solve(x)              # Inverse matrix of x
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5

> det(x)                # A determinant of x
[1] -2
```

06 Array: A useful function

Function useful for array operation

- apply function: Applying functions by row or column in an array

```
> x = array(1:12, c(3, 4))
> x
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

> apply(x, 1, mean)
[1] 5.5 6.5 7.5

> apply(x, 2, mean)
[1]  2  5  8 11
```

Matrix data

Middle value margin

Function to operate

 # If the middle value is 1, apply the function by row

 # If the middle value is 2, apply the function by column

- dim function: Size of the array (number of dimensions)

```
> x = array(1:12, c(3, 4))
> dim(x)
[1] 3 4
```

06 Array: A useful function

- sample function: Sampling from a vector or array.

```
> x = array(1:12, c(3, 4))

> sample(x)                                   # Extract by randomly mixing array elements
 [1]  7  8  1 10  6  4  2  9 12  3  5 11

> sample(x, 10)                               # Extract 10 of the array elements
 [1]  2  3 12  9  4  7  8 11  5  6

> sample(x, 10, prob = c(1:12)/24)            # Extraction probability of each element is
 [1] 11  2 12  9 10  6  7  8  5  1                  available to set differently

> sample(10)                                  # Samples can be created using simple numbers
 [1]  9  3 10  4  2  5  8  7  1  6
```

07 Data Frame

Data Frame

- It has the most commonly used structure of a table.
- Unlike a matrix, you can store a mix of different data types.
- Unlike the list, the number of rows must be matched and saved.

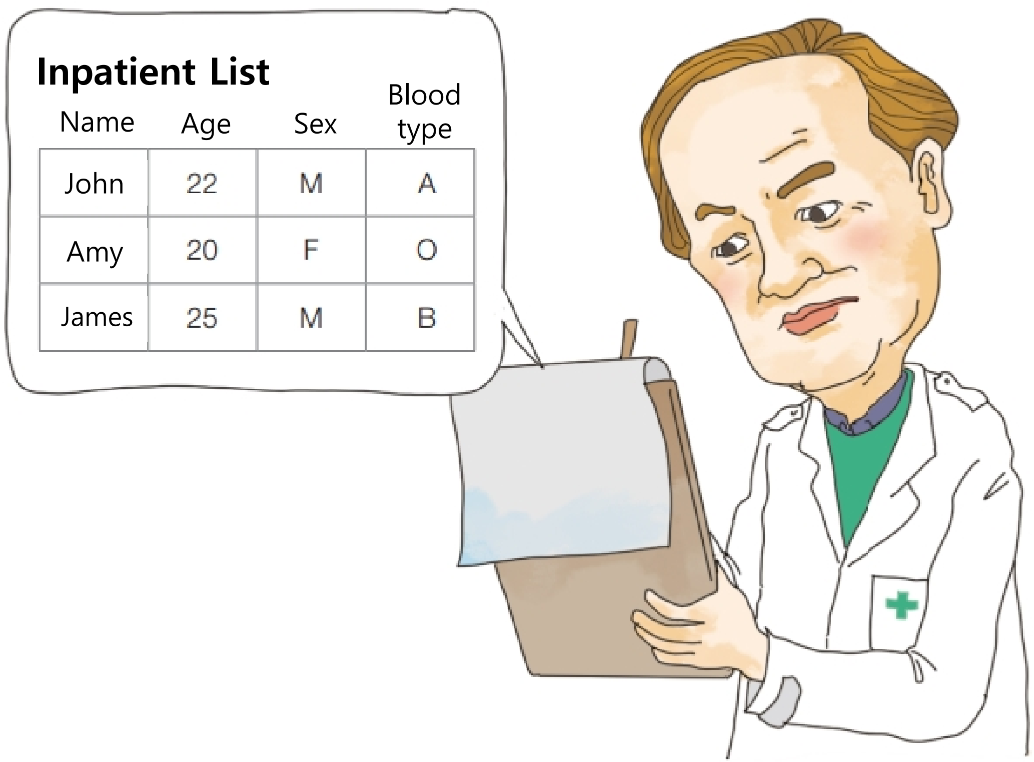


Figure 3-5 **Configurations of Data Frame**

07 Data Frame

- Data Frame generation: Using data.frame function

```
> name = c("John", "Amy", "James")
> age = c(22, 20, 25)
> gender = factor(c("M", "F", "M"))
> blood.type = factor(c("A", "O", "B"))

> patients = data.frame(name, age, gender, blood.type)
> patients
   name age gender blood.type
1 John   22      M          A
2 Amy    20      F          O
3 James  25      M          B
```

07 Data Frame

- Access to data frame elements: using such as conditional expression, $, [,]

```
> patients$name        # Output of name property value
[1] John Amy James
Levels: Amy James John

> patients[1, ]        # Output of 1-row values
  name age gender blood.type
 1 John  22      M          A

> patients[, 2]        # Output of 2-row values
[1] 22 20 25

> patients[3, 1]        # Output of 3-row 1-column values
[1] John
Levels: John James Amy
> patients[patients$name== "John", ]  # Extracting information about John among patients
  name age gender blood.type
1 John  22      M          A

> patients[patients$name== "John", c("name", "age")]  # Extract only name and age
  name age
1 John  22
```

07 Data Frame : Useful functions

Functions useful for Data Frame

- attach・detach function: Rename property name of data frame to variable name

```
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10

> speed
Error: object 'speed' not found

> attach(cars)

> speed
 [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13 14 14 14 14 15
15 15 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 20 22 23 24 24 24 24 25

> detach(cars)

> speed
Error: object 'speed' not found
```

The basic function of the head function is to extract 6 data from the front

❶ Occur because the Speed variable does not exist independently

Each attribute of 'cars' is available as a variable through 'attach function'

The variable name, speed, is directly available

That using cars's attribute as a variable is available to cancel through 'detach function'.

Unable to access the 'speed' variable any longer.

❷ After running 'detach',
the attribute 'speed' of 'cars' is no longer available outside

07 Data Frame : Useful functions

- with function: Applying various functions to the Data Frame

```
>  # Applying function using data properties
> mean(cars$speed)
[1] 15.4
> max(cars$speed)
[1] 25

>  # Applying function using with function
> with(cars, mean(speed))
[1] 15.4
> with(cars, max(speed))
[1] 25
```

07 Data Frame : Useful functions

- subset function: Extract only some data from the data frame

```
> # Extract only data with a speed greater than 20
> subset(cars, speed > 20)
   speed dist
44    22   66
45    23   54
46    24   70
47    24   92
48    24   93
49    24  120
50    25   85
```

```
> # Extract only data with a speed greater than 20 except dist
> subset(cars, speed > 20, select = -c(dist))
   speed
44    22
45    23
46    24
47    24
48    24
49    24
50    25
```

```
> # Extract only dist-data with a speed greater than 20
> subset(cars, speed > 20, select = c(dist))
   dist
44   66
45   54
46   70
47   92
48   93
49  120
50   85
```

To select multiple columns, divide within c( ) by using ' , '

07 Data Frame : Useful functions

- na.omit function: Remove missing-values(NA) from Data Frame

```
> head(airquality)                    # airquality-data contains NA
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6

> head(na.omit(airquality))           # Extracts values except for NA
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
7    23     299  8.6   65     5   7
8    19      99 13.8   59     5   8
```

07 Data Frame : Useful functions

- merge function: Merge multiple Data Frames

```
> name = C("John", "Amy", "James")
> age = c(22, 20, 25)
> gender = factor(c("M", "F", "M"))
> blood.type = factor(c("A", "O", "B"))

> patients1 = data.frame(name, age, gender)
> patients1
   name age gender
1  John  22      M
2  Amy   20      F
3 James  25      M

> patients2 = data.frame(name, blood.type)
> patients2
   name blood.type
1  John          A
2  Amy           O
3 James          B
```

```
> patients = merge(patients1, patients2, by = "name")
> patients
   name age gender blood.type
1 James  25      M          B
2  John  22      M          A
3   Amy  20      F          O
```

08 List

List

- It can include data structures that have different base data types each other.
- A group of data in a broader meaning than a data frame.
- Unlike data frames, all properties don't have to be the same size.

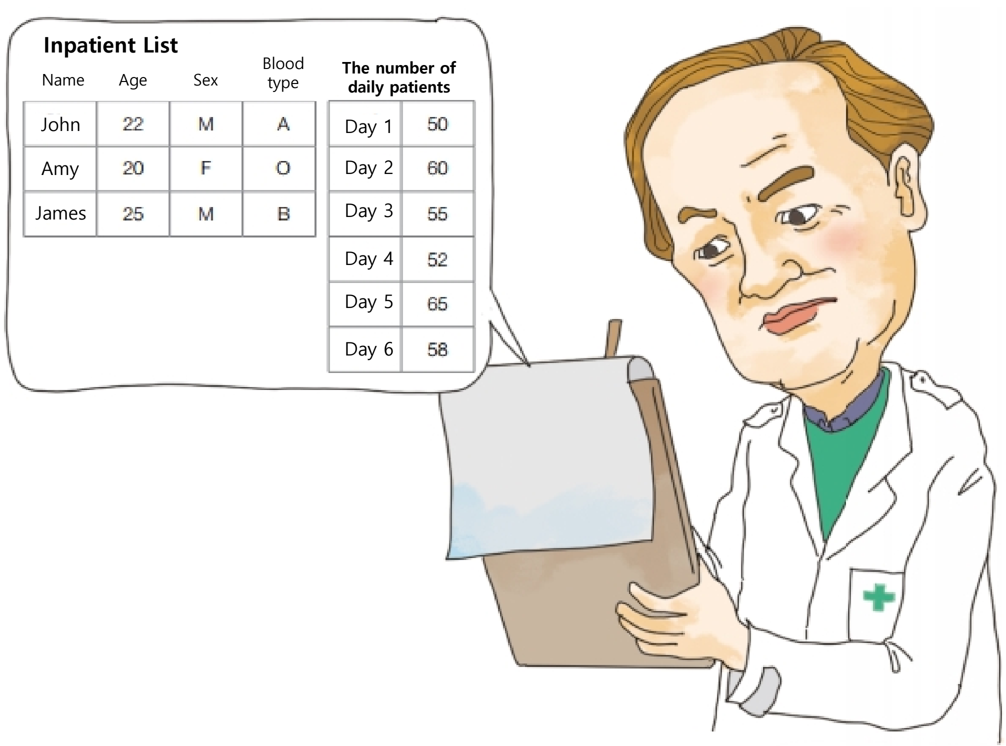


Figure 3-6 **Configurations of List**

08 List

List generation: Using list function

```
> patients = data.frame(name = c("John", "Amy", "James"), age = c(22, 50, 25),
gender = factor(c("M", "F", "M")), blood.type = factor(c("A", "O", "B")))
> no.patients = data.frame(day = c(1:6), no = c(50, 60, 55, 52, 65, 58))

> # Simply add data
> listPatients = list(patients, no.patients)
> listPatients
[[1]]
  name age gender blood.type
1 John  22      M          A
2 Amy   20      F          O
3 James 25      M          B

[[2]]
  day no
1   1 50
2   2 60
3   3 55
4   4 52
5   5 65
6   6 58
```

```
> # Add data by naming each data
> listPatients = list(patients=patients, no.patients = no.patients)
> listPatients
$patients
  name age gender blood.type
1 John  22      M          A
2 Amy   20      F          O
3 James 25      M          B

$no.patients
  day no
1   1 50
2   2 60
3   3 55
4   4 52
5   5 65
6   6 58
```

08 List

- Access to list elements : using $, [[]]

```
> listPatients$patients                # Enter element name
   name age gender blood.type
1  John  22      M          A
2   Amy  20      F          O
3 James  25      M          B

> listPatients[[1]]                    # Enter an index
   name age gender blood.type
1  John  22      M          A
2   Amy  20      F          O
3 James  25      M          B

> listPatients[["patients"]]           # Enter element name in “”
   name age gender blood.type
1  John  22      M          A
2   Amy  20      F          O
3 James  25      M          B

> listPatients[["no.patients"]]        # Enter element name in “”
  day no
1   1 50
2   2 60
3   3 55
4   4 52
5   5 65
6   6 58
```

08 List

Functions useful for List

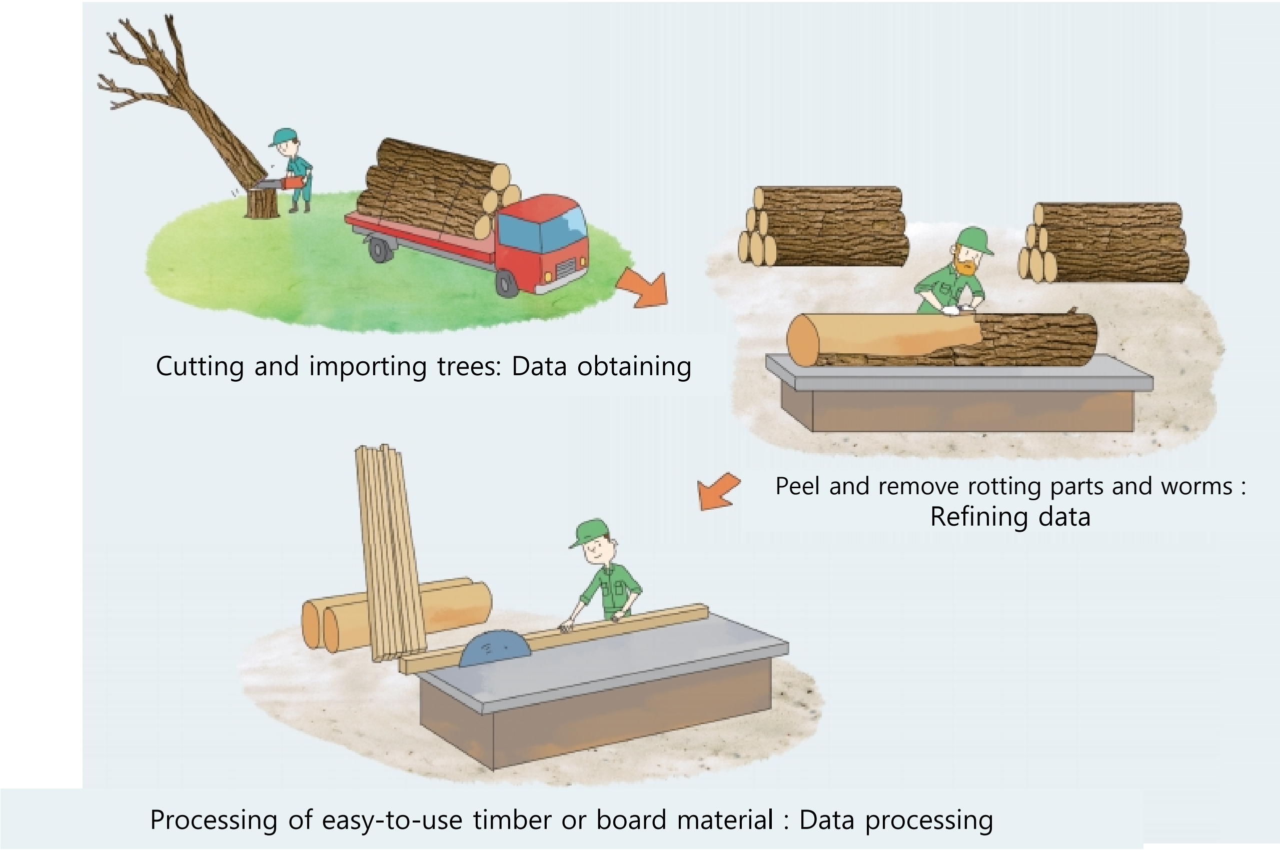
- lapply · sapply function: Applying various functions to list elements

```
> # no.patients    # getting an average of elements
> lapply(listPatients$no.patients, mean)
$day
[1] 3.5

$no
[1] 56.66667

> # getting an average of 'patients' elements. Non-numeric forms cannot be obtained average
> lapply(listPatients$patients, mean)
$name
[1] NA

$age
[1] 22.33333

$gender
[1] NA

$blood.type
[1] NA
```

```
> sapply(listPatients$no.patients, mean)
     day       no
 3.50000 56.66667

># If 'simplify option' in sapply() is set to F, it returns the same result as the result in lapply()
> sapply(listPatients$no.patients, mean, simplify = F)
$day
[1] 3.5

$no
[1] 56.66667
```

Preview

- Data collection and refining
  - Data can be obtained through Internet surfing, documentation, surveys or experiments.
  - Collected data should be refined appropriately to be used for data science
  - Most data processing can be done using refined data..



01 Read and write files

- Most of the data exist in file form.
- File read and write functions provided by R

Table 4-1 **File read and write functions available in R**

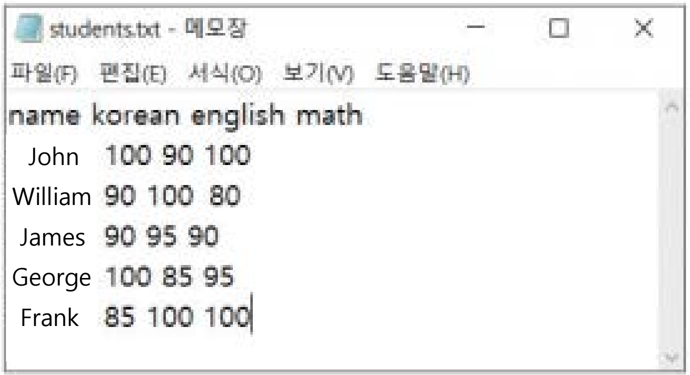
Package	Function
Base(basic) Package	scan, write, write.table, read.table, save, load, write.csv, read.csv, etc.
readr Package	write_csv, read_csv
data.table Package	fwrite, fread
feather Package	write_feather, read_feather

01 Read and write files : Read the file

Read the file

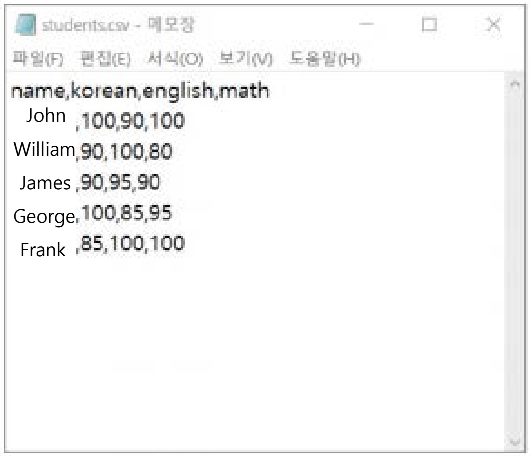
- read.table function : Use to read plain text files

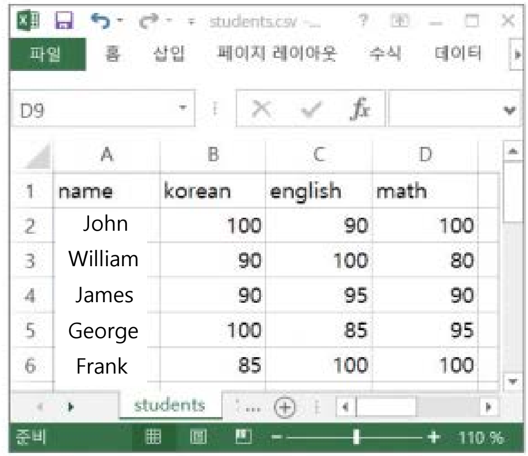
```
> students=read.table("C:/Sources/students.txt", header=T)
> students
     name korean english math
1    John    100      90  100
2 William     90     100   80
3   James     90      95   90
4  George    100      85   95
5   Frank     85     100  100

>  # Check the structure of a read file
> str(students)
'data.frame':        5 obs. of  4 variables:
 $ name   : Factor w/ 5 levels "John", "William",,..: 1 2 3 4 5
 $ korean : int  100 90 90 100 85
 $ english: int  90 100 95 85 100
 $ math   : int  100 80 90 95 100
```



01 Read and write files : Read the file

- read.csv function: Used to read CSV(Comma-Separated Values)





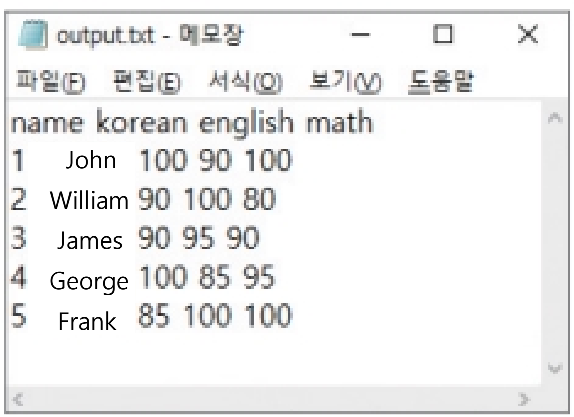

```
# Do not need to specify the header option because the first row is a header
> students=read.csv("C:/Sources/students.csv")
> students
  name    korean  english  math
1   John     100       90   100
2 William     90      100    80
3  James      90       95    90
4 George     100       85    95
5  Frank      85      100   100
```

01 Read and write files : Write the file

Write the file

- write.table function: Used to save as a plain text file

```
> students=read.table("C:/Sources/students.txt", header=T, as.is=T)

# A double quotation mark is shown in the string
> write.table(students, file="C:/Sources/output.txt")

# No double quotation marks are shown in the string.
> write.table(students, file="C:/Sources/output.txt", quote=F)
```




```
name korean english math
1 John 100 90 100
2 William 90 100 80
3 James 90 95 90
4 George 100 85 95
5 Frank 85 100 100
```

Figure 4-4 The file saved by applying quote=F with write.table function

01 Read and write files : Write the file

- write.csv function: Used to save as CSV file

```
> write.csv(students, file="C:/Sources/output.csv", quote=F)
```

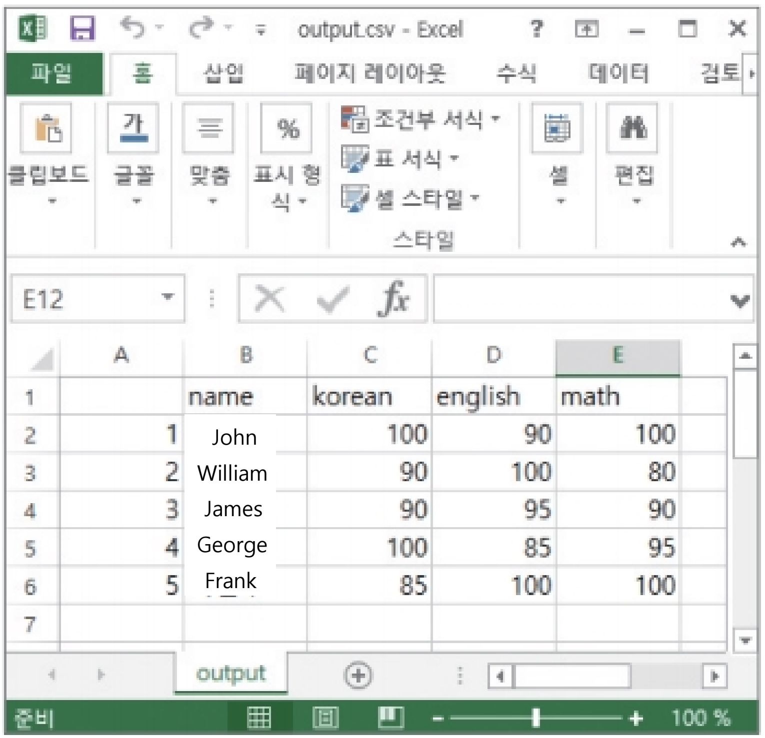


Figure 4-4 **The file to save with write.csv function**

02 IF statement and Iteration statement for data refining: IF statement

- It is possible to work for a variety of purposes, such as to find values that meet certain conditions or to extract and compute values from some sections for data refining.
- Let's take a look at the function of the conditional find provided by R, and learn about IF statement and Iteration statement and how to use it.

- **Table 4-2 IF statement format**

Way to extract elements that meet the conditions	Type
Specifying row/column conditions in [ ]	Variable name [row conditional statement, column conditional statement]
Utilizing IF statements	If (conditional statement) expression
Utilizing IF-ELSE statements	If-Else (conditional statement, Return value if True, Return value if False)

Specifying row/column conditions in [ ]

- In the case of vector

```
> test=c(15, 20, 30, NA, 45)     # In the case of vector
> test[test<40]                  # Extracting elements with values below 40
[1] 15 20 30 NA

> test[test%%3!=0]               # Extracting elements whose values cannot be divisible by 3
[1] 20 NA

> test[is.na(test)]              # Extracting elements with NA
[1] NA

> test[!is.na(test)]             # Extracting elements with not NA
[1] 15 20 30 45

> test[test%%2==0&!is.na(test)]  # Extracting elements that are multiples of 2
[1] 20 30                          and are not NA
```

02 IF statement and Iteration statement for data refining: IF statement

- Specifying row/column conditions in [ ]
  - In the case of DataFrame

```
> characters=data.frame(name= C("John", "Amy", "James") ,   age=c(30, 16, 21),
gender=factor(c("M", "F","M")))              # In the case of DataFrame
> characters
   name age gender
1  John 30      M
2   Amy 16      F
3 James 21      M

> characters[characters$gender=="F", ]    # Extracting rows of female
   name age gender
2   Amy 16      F
                              # Extracting the rows of men under 30 years of age
> characters[characters$age<30&characters$gender=="M", ]
   name age gender
3 James 21      M
```

02 IF statement and Iteration statement for data refining: IF statement

Using If statement (if, else if, else)

- In case two condition branches are required

```
> x = 5
> if(x %% 2==0) {
+         print("x is an even number")    # Performing when the conditional statement is true
+ }     else {
+         print("x is an odd number")     # Performing when the conditional statement is false
+ }
[1]  "x is an odd number"
```

- In case three condition branches are required

```
> x = -1
> if(x>0) {
+       print('x is a positive value.')   # Output if x is greater than zero
+ } else if(x<0) {
+       print('x is a negative value.')   # Output if x is less than 0 without meeting the above conditions
+ } else {
+       print('x is zero.')               # Output if all of the above conditions are not met
+ }
[1] "x is a negative value."
```

02 IF statement and Iteration statement for data refining: IF statement

Using If-else statement

- The form of the if/else statement combined
- Directions: ifelse(if statement, Return-value if the conditional statement is True,
  Return-value if the conditional statement is Flase)

```
> x=c(-5:5)
> options(digits=3)  # Setting the effective digit to three digits when expressing a number
> sqrt(x)
[1]  NaN  NaN  NaN  NaN  NaN 0.00 1.00 1.41 1.73 2.00 2.24
Warning message:
In sqrt(x) : NaNs produced

                                  # To prevent NaN occurring,
> sqrt(ifelse(x>=0, x, NA))    if the value is a negative number, marking as NA
[1]   NA   NA   NA   NA   NA 0.00 1.00 1.41 1.73 2.00 2.24
```

02 IF statement and Iteration statement for data refining: IF statement

- Ex) processing the conditional statement after data is read from a file
  - A program that is treated with NA if a score other than 0-100 is entered.

```
> students=read.csv("C:/Sources/students.csv")
> students                    # Data contains values above 100 and negative values
     name korean english math
1    John    100      90  100
2 William     90     120   80
3   James     90      95   90
4  George    100      85 -100
5   Frank     85     100  100

> students[, 2]=ifelse(students[, 2]>=0 & students[, 2]<=100, students[, 2], NA)
> students[, 3]=ifelse(students[, 3]>=0 & students[, 3]<=100, students[, 3], NA)
> students[, 4]=ifelse(students[, 4]>=0 & students[, 4]<=100, students[, 4], NA)
> students                    # Any value other than 0-100 of the values in columns 2 to 4
                              are treated as NA With the if-else statement
     name korean english math
1    John    100      90  100
2 William     90      NA   80
3   James     90      95   90
4  George    100      85   NA
5   Frank     85     100  100
```

02 IF statement and Iteration statement for data refining: Iteration statement

Iteration statement

- There are times when data reviews require repeated changes to values. For example, a case of compare rows 0 through 10 of the data frame.
- Iteration statements provided by R include 'repeat', 'while', 'for'

Table 4-3 **Iteration statement format**

Iteration statement	meaning

`repeat {`
A sentence to repeat
`}`
	Repeating the sentence in the block.

`while(If statement) {`
A sentence to be performed when the conditional statement is true
`}`
	Repeating the sentence in the block when the conditional statement is true.

`for(variable in data) {`
A sentence to repeat
`}`
	Each element of the data is assigned to a variable, while each performs a sentence within the block.

02 IF statement and Iteration statement for data refining: Iteration statement

Using repeat statement

- Increasing the number from 1 to 10 by 1

```
> # Increasing number from 1 to 10 using repeat statement
> i=1                        # The starting value of i is 1
> repeat {
+      if(i>10) {            # If i is above 10, stop(break) it from repeating
+            break
+      } else {
+            print(i)
+            i=i+1           # Increasing i by 1
+      }
+   }
[1] 1
[1] 2
· · · skip ·
 · ·
[1] 9
[1] 10
```

Using while statement

- Increasing the number from 1 to 10 by 1

```
> i=1                    # The starting value of i is 1
> while(i<10) {          # Repeating while i is 10 or less
+       print(i)
+       i=i+1            # Increasing i by 1
+ }
[1] 1
 · · · skip ·
   · ·
[1] 10
```

-

```
> i=1
> while(i<10) {
+       print(paste(2, "X", i, "=", 2*i))
+       i=i+1
+ }
[1] "2 X 1 = 2"
 · · · skip ·
   · ·
[1] "2 X 9 = 18"
```

02 IF statement and Iteration statement for data refining: Iteration statement

Using for statement

- Increasing the number from 1 to 10 by 1

```
> for(i in 1:10) {
+       print(i)
+ }
[1] 1
 · · · skip ·
      · ·
[1] 10
```

-

```
> for(i in 2:9) {
+       for(j in 1:9) {
+             print(paste(i, "X", j, "=", i*j))
+       }
+ }
```

02 IF statement and Iteration statement for data refining

- Ex) Use conditional and iterative statements to find values within a certain range that fit the condition.

```
# Output only an even number of numbers from 1 to 10
> for(i in 1:10) {
+     if(i%%2==0) {
+         print(i)
+     }
+ }
[1] 2
[1] 4
[1] 6
[1] 8
[1] 10
```

```
# Output only prime number of the numbers from 1 to 10
> for(i in 1:10) {
+     check=0
+     for(j in 1:i) {
+         if(i%%j==0) {
+             check=check+1
+         }
+     }
+     if(check==2) {
+         print(i)
+     }
+ }
[1] 2
[1] 3
[1] 5
[1] 7
```

02 IF statement and Iteration statement for data refining

- Ex) Processing iteration statement and condition statement after reading data from file

```
> students=read.csv("C:/Sources/students.csv")
> students        # Data contains values above 100 and negative values
     name korean english math
1    John    100      90  100
2 William     90     120   80
3   James     90      95   90
4  George    100      85 -100
5   Frank     85     100  100

> for(i in 2:4) {
+   students[,i]=ifelse(students[, i]>=0&students[,i]<=100, students[,i], NA)
+ }

> students          # Using if-else statement to treat values other than 0 to 100
     name korean english math                      of the 2nd to 4th columns as NA
1    John    100      90  100
2 William     90      NA   80
3   James     90      95   90
4  George    100      85   NA
5   Frank     85     100  100
```

03 User-defined function: Grouping desired functions

Function

- The relational expression between input and output can be called a function.
- Let's create a variety of functions to suit the user's purpose.

Structure of user-defined functions

```
Function name = function ( factor1, factor2, ･ ･ ･ )
{
+     Code to perform when function operates
+     return(return value)
}
```

03 User-defined function: Grouping desired functions

- Ex) Function to obtain a Factorial

```
> fact= function(x) {          # The function name is fact, input is x
+     fa = 1                   # A variable to store the factorial value
+     while(x>1) {             # Repeat while x is greater than 1
+         fa = fa*x            # Multiply the x value by fa and store it back in fa
+          x = x-1             # Reduce the x value by 1
+     }
+     return(fa)               # Return the final calculated value of the fa
+ }
> fact(5)                      # Output after calculating 5!
[1] 120
```

04 Example of data refining 1: Processing of missing values

- The data we collect may have missing values.
- Missing values are those that are intentionally or accidentally omitted from the data.
- Proper processing is required during the refining process since processing data while leaving missing values intact can cause errors in the results or incorrect operation.

Table 4-4 Method of processing missing values

Method	Function
Using is.na function	If there is NA data, indicate T, if not F.
Using na.omit function	Remove data that is NA. That is, clear the row containing NA.
Using the properties of a function	When performing a function with na.rm=T, exclude NA.

04 Example of data refining 1: Processing of missing values

Using is.na function

- Ex) Processing missing values in airquality data

```
# There is a total of 44 NA
> table(is.na(airquality))
FALSE  TRUE
  874    44

# Temp has confirmed that there is no NA
> table(is.na(airquality$Temp))
FALSE
  153

# Ozone has a total of 37 NA
> table(is.na(airquality$Ozone))
FALSE  TRUE
  116    37

# Temp without NA can get the value of average
> mean(airquality$Temp)
[1] 77.88235

# Ozone with NA value appears NA
> mean(airquality$Ozone)
[1] NA
```

```
# Extracting only value without NA from Ozone property
> air_narm=airquality[!is.na(airquality$Ozone), ]
> air_narm
    Ozone Solar.R Wind Temp Month Day
1      41     190  7.4   67     5   1
2      36     118  8.0   72     5   2
3      12     149 12.6   74     5   3
… 생략 …
149    30     193  6.9   70     9  26
151    14     191 14.3   75     9  28
152    18     131  8.0   76     9  29
153    20     223 11.5   68     9  30

# mean function operates normally in data in which missing values have been removed
> mean(air_narm$Ozone)
[1] 42.12931
```

04 Example of data refining 1: Processing of missing values

- Using na.omit function
  - Ex) Processing missing values from airquality data

```
# Processing missing values using na.omit function
> air_narm1=na.omit(airquality)
> mean(air_narm1$Ozone)
[1] 42.0991
```

If the missing values for the ozone are removed, the output is 42.12931
That is, after performing the first line as air_narm1 = na.omit (airquality$Ozone), you can perform mean(air_narm1)

- Setting function property, Na.rm to TRUE
  - Ex) processing missing values in airquality data

```
# Process missing values using function attribute na.rm
> mean(airquality$Ozone, na.rm=T)
[1] 42.12931
```

05 Example of Data Refinement 2: Outlier processing

- In addition to missing values, data may contain logical or statistically unusual data. These data are called outliers.
- "In statistics, the outlier is an observation value far away from other observation values."
- Ex)Let's deal with obvious outliers. Processing of outlier values of gender and blood type.
  - Let's assume that only M and F exist in gender, and blood types are expressed only as A, B, O, and AB.

```
> # Patient data with outliers
> patients=data.frame(name=C("patient1", "patient2", "patient3", "patient4", "patient5"), age=
c(22, 20, 25, 30, 27), gender=factor(c("M", "F", "M", "K", "F")), blood.type=
factor(c("A", "O", "B", "AB", "C")))
> patients
      name age gender blood.type
1 patient1  22      M          A
2 patient2  20      F          O
3 patient3  25      M          B
4 patient4  30      K         AB
5 patient5  27      F          C
```

05 Example of Data Refinement 2: Outlier processing

- K entered for gender or C entered for blood type are clearly an outlier.

```
# Remove outlier from gender
> patients_outrm=patients[patients$gender=="M"|patients$gender=="F", ]
> patients_outrm
     name age gender blood.type
1 patient1  22      M          A
2 patient2  20      F          O
3 patient3  25      M          B
5 patient5  27      F          C
```

```
# Remove outlier from gender and blood type
> patients_outrm1=patients[(patients$gender=="M"|patients$gender=="F") &
(patients$blood.type=="A"|patients$blood.type=="B"|patients$blood.type==
"O"|patients$blood.type=="AB"), ]
> patients_outrm1
     name age gender blood.type
1 patient1  22      M          A
2 patient2  20      F          O
3 patient3  25      M          B
```

05 Example of Data Refinement 2: Outlier processing

- If you express all the outliers in NA, you will be able to use the related functions with NA covered in Section 04.

- Ex) Processing of outlier values of gender and blood type
  - Expressing as 1 for male and 2 for female in gender.
  - Expressing as 1, 2, 3, 4 respectively for A, B, O, AB blood types

```
# Patient data with outliers
> patients=data.frame(name=C("patient1", "patient2", "patient3", "patient4", "patient5"), age=
c(22, 20, 25, 30, 27), gender=c(1, 2, 1, 3, 2), blood.type=c(1, 3, 2, 4, 5))
> patients
      name age gender blood.type
1 patient1  22      1          1
2 patient2  20      2          3
3 patient3  25      1          2
4 patient4  30      3          4
5 patient5  27      2          5
```

05 Example of Data Refinement 2: Outlier processing

- Change the outlier in gender to missing value(NA)

```
> patients$gender=ifelse((patients$gender<1|patients$gender>2), NA, patients$gender)
> patients
     name age gender blood.type
1 patient1  22      1          1
2 patient2  20      2          3
3 patient3  25      1          2
4 patient4  30     NA          4
5 patient5  27      2          5
```

- Change the outlier in blood type to missing value(NA)

```
> patients$blood.type=ifelse((patients$blood.type<1|patients$blood.type>4),
NA, patients$blood.type)
> patients
     name age gender blood.type
1 patient1  22      1          1
2 patient2  20      2          3
3 patient3  25      1          2
4 patient4  30     NA          4
5 patient5  27      2         NA
```

05 Example of Data Refinement 2: Outlier processing

- Remove all outliers expressed as missing values

```
> patients[!is.na(patients$gender)&!is.na(patients$blood.type), ]
     name age gender blood.type
1 patient1  22      1          1
2 patient2  20      2          3
3 patient3  25      1          2
```

05 Example of Data Refinement 2: Outlier processing

- Let's use more real data to process with outliers.
- In real-life situations, it is often ambiguous to define outliers. For example, if a person's age is input to be 120 years old, it is difficult to clearly determine whether this is an outlier or a normal value. A 200-year-old makes it easier to decide.
- Here is an example of using boxplot to distinguish between normal and outlier values.

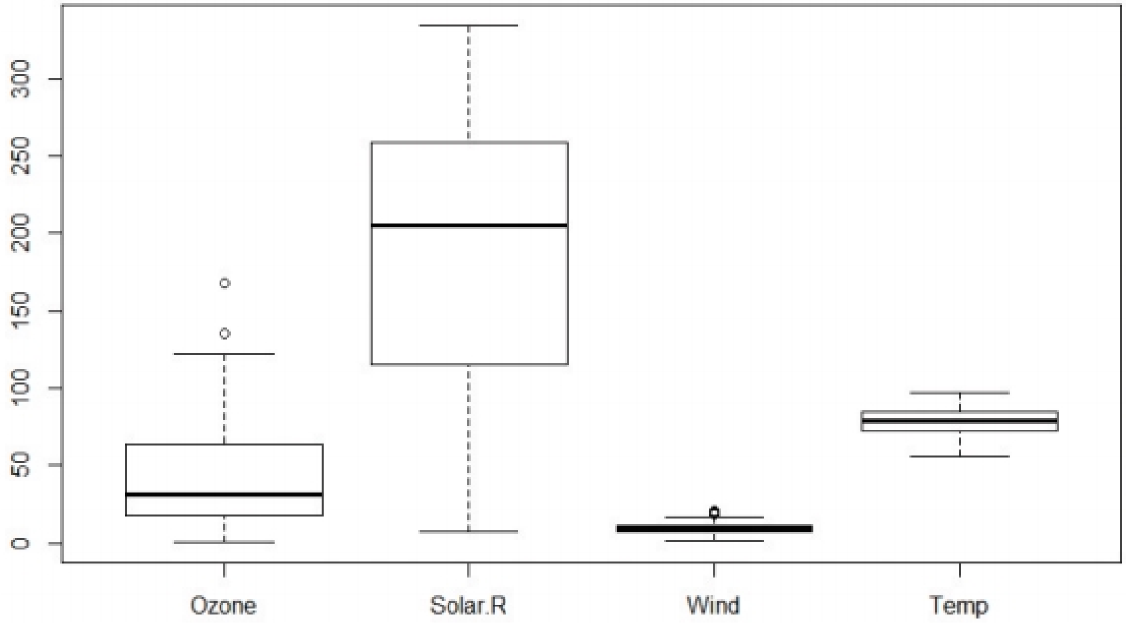


Figure 4-7 **Airquality data drawn using boxplot**

05 Example of Data Refinement 2: Outlier processing

- Ex) Processing outliers of airquality data
  - Using boxplot to classify outlier

```
> boxplot(airquality[, c(1:4)])      # boxplot for Ozone, Solar.R, Wind, Temp (Figure 4-7)
> boxplot(airquality[, 1])$stats     # Calculate boxplot statistics-value for ozone
      [,1]
[1,]   1.0    → Values below this value can be classified as an outlier
[2,]  18.0
[3,]  31.5
[4,]  63.5
[5,] 122.0    → Values exceeding this value can be classified as an outlier
attr(,"class")
        1
"integer"

> air=airquality                 # Copy airquality data to the temporary storage variable
> table(is.na(air$Ozone))        # Checking the current NA count of Ozone
FALSE  TRUE
  116    37
```

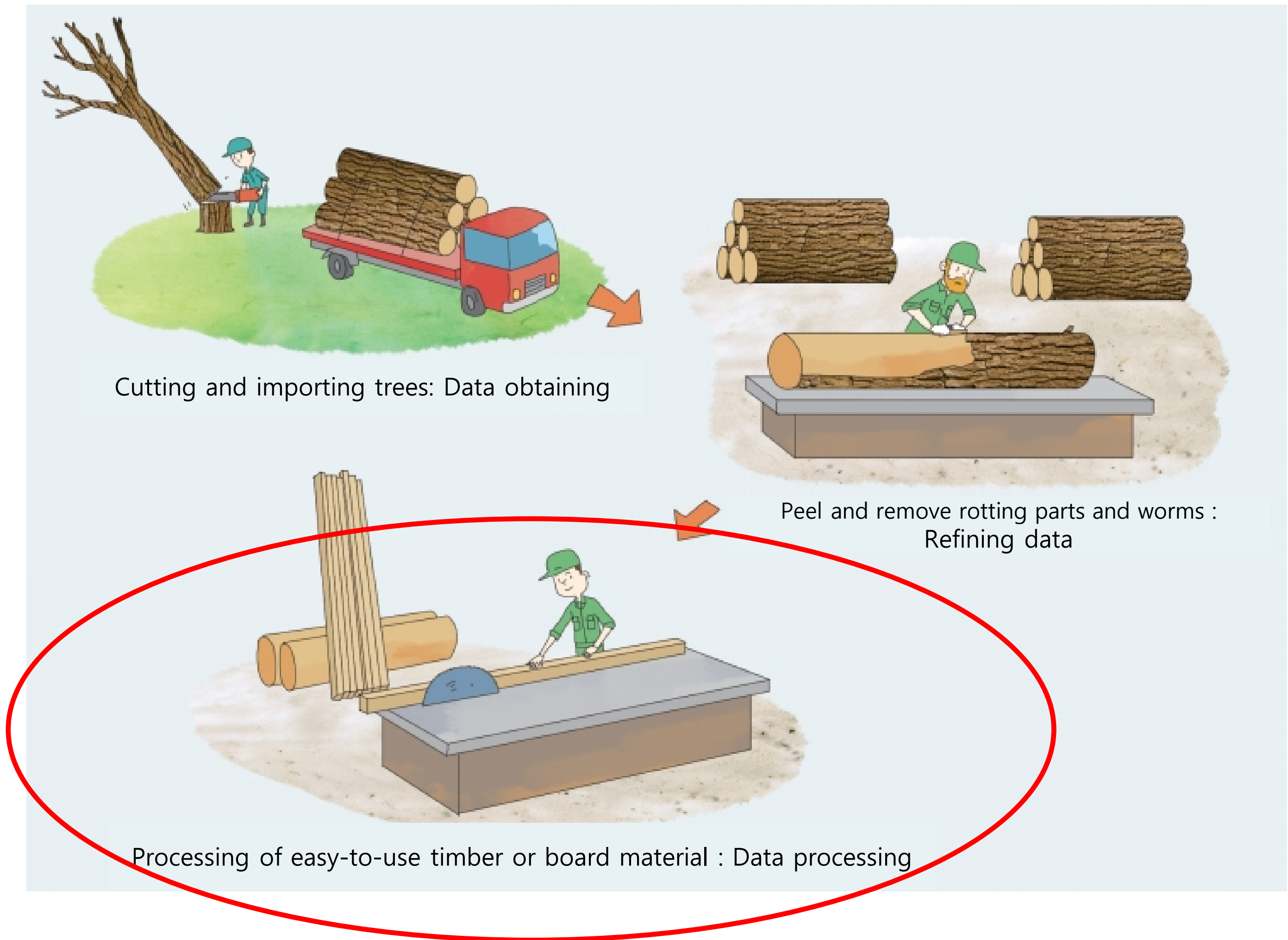
05 Example of Data Refinement 2: Outlier processing

- Remove NA after processing with an outlier

```
# Change outlier to NA
> air$Ozone =ifelse(air$Ozone<1|air$Ozone>122, NA, air$Ozone)
> table(is.na(air$Ozon        # Checking the number of NA after processing of outlier
FALSE  TRUE                                (increased by 2)
  114    39

# Removing NA
> air_narm =air[!is.na(air$Ozone), ]
> mean(air_narm$Ozone)     # By eliminating two outliers, the value was further reduced than
[1] 40.21053                        the results using is.na function
```

PREVIEW

- Well-refined data is of great value, but unrefined data can difficult to extract meaning as well as lead to incorrect conclusions.



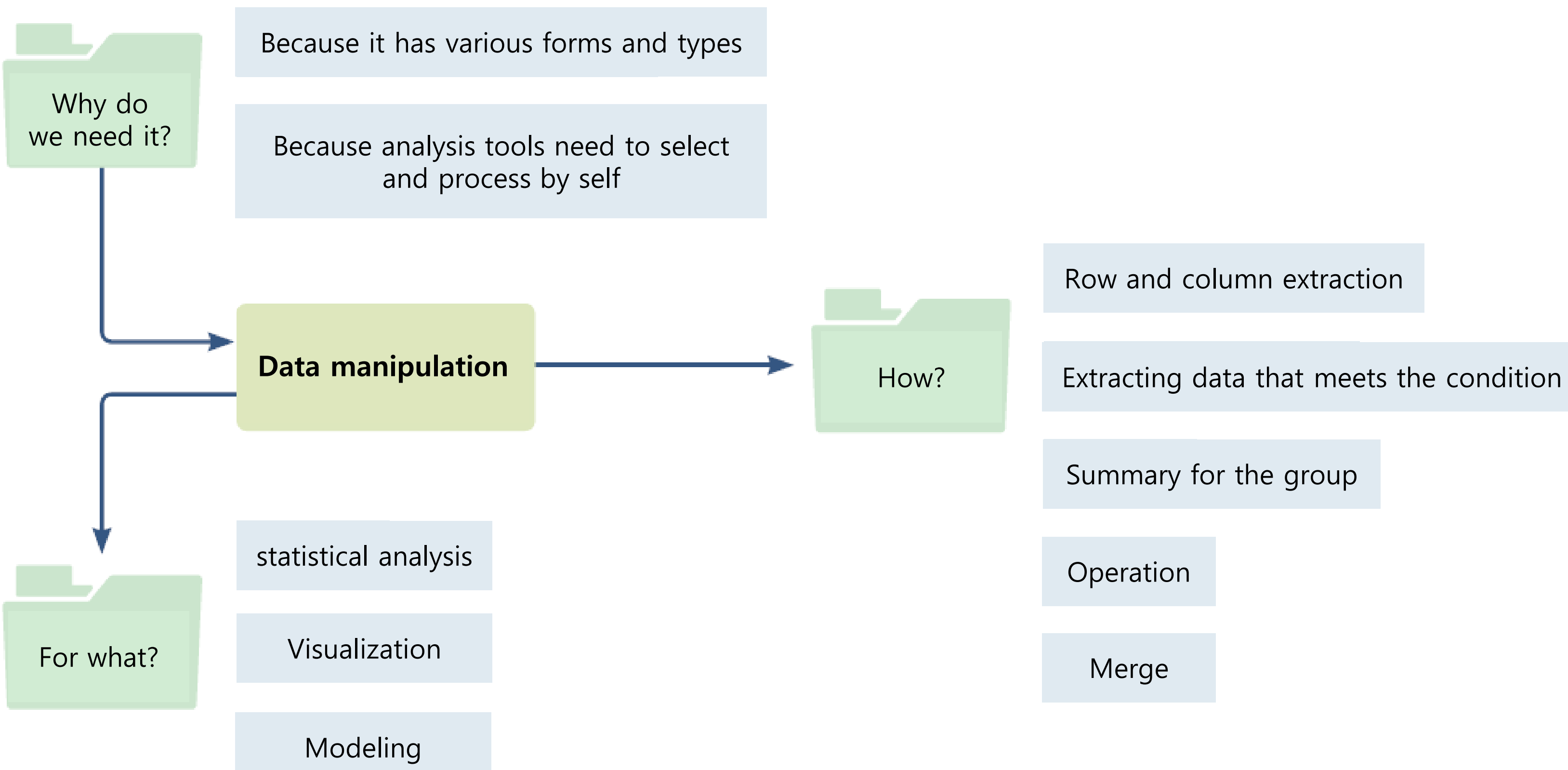
01 What is Data manipulation?

- In that it is done with a broad and specific purpose, data wrangling is different from refining that removes unnecessary elements of data and makes it easier to use.

- Requires proper data wrangling in almost every field.
  - Statistic analysis to find meaning from data
  - Visualization for effective observation
  - Modeling for estimating causality, etc.

01 What is Data manipulation?

- Data manipulation is that transform data in order to analyze it more effectively. data wrangling
- Processing is available easily and quickly using digitized data and analytical tools such as R.



02 Data manipulation using base R

gapminder library

- It contains part of a set of Gapminder data that aggregates life expectancy (LifeExp), gross domestic product (gdpPercap) and population (pop) data from around the world

Table 5-1 Configuration items in the gapminder data frame

Column name (variable name)	Variable type	content
country	Factor with 142 levels	Country name
continent	Factor with 5 levels	The continent to which the country belongs
year	int	Observation year in 1952-2007 (in five-year increments)
lifeExp	num	Life expectancy
pop	int	Population
gdpPercap	num	Gross national product per capita (considering the price increase rate)

02 Data manipulation using base R

- gapminder library

```
> library(gapminder)
> library(dplyr)
> glimpse(gapminder)
Observations: 1,704
Variables: 6
$ country   <fct> Afghanistan, Afghanistan, Afghanistan, Afghanistan,
Afghanistan, Afghanistan, Afghanist...
$ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia,
Asia, Asia, Europe, Europe,...
$ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997,
2002, 2007, 1952, 1957, 196...
$ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 40.822,
41.674, 41.763, 42.129,...
$ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372,
12881816, 13867957, 16317921,...
$ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.1134,
978.0114, 852.3959, 649.341...
```

02 Data manipulation using base R

Extracting of samples and attributes

- The life expectancy(lifeExp) of each county(country)

```
> gapminder[, c("country", "lifeExp")]
# A tibble: 1,704 x 2
   country     lifeExp
   <fct>         <dbl>
 1 Afghanistan    28.8
 2 Afghanistan    30.3
 3 Afghanistan    32.0
 4 Afghanistan    34.0
 5 Afghanistan    36.1
 6 Afghanistan    38.4
 7 Afghanistan    39.9
 8 Afghanistan    40.8
 9 Afghanistan    41.7
10 Afghanistan    41.8
# ... with 1,694 more rows
```

Gapminder data records samples measured every five years for each country, so shown multiple life expectancy values outputs in the same country

→ It is more appropriate to view the year of measurement together.

02 Data manipulation using base R

Extracting of samples and attributes

- The life expectancy(lifeExp) of each county(country) + measurement year (year)

```
> gapminder[, c("country", "lifeExp", "year")]
# A tibble: 1,704 x 3
   country     lifeExp  year
   <fct>         <dbl> <int>
 1 Afghanistan    28.8  1952
 2 Afghanistan    30.3  1957
 3 Afghanistan    32.0  1962
 4 Afghanistan    34.0  1967
 5 Afghanistan    36.1  1972
 6 Afghanistan    38.4  1977
 7 Afghanistan    39.9  1982
 8 Afghanistan    40.8  1987
 9 Afghanistan    41.7  1992
10 Afghanistan    41.8  1997
# ... with 1,694 more rows
```

02 Data manipulation using base R

Extracting of samples and attributes

- Rows often do not have names, so it is common to specify row numbers or use conditional expressions as follows.

```
> gapminder[1:15, ]
# A tibble: 15 x 6
   country     continent  year lifeExp      pop gdpPercap
   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
 1 Afghanistan Asia       1952    28.8  8425333      779.
 2 Afghanistan Asia       1957    30.3  9240934      821.
 3 Afghanistan Asia       1962    32.0 10267083      853.
 4 Afghanistan Asia       1967    34.0 11537966      836.
 5 Afghanistan Asia       1972    36.1 13079460      740.
 6 Afghanistan Asia       1977    38.4 14880372      786.
 7 Afghanistan Asia       1982    39.9 12881816      978.
 8 Afghanistan Asia       1987    40.8 13867957      852.
 9 Afghanistan Asia       1992    41.7 16317921      649.
10 Afghanistan Asia       1997    41.8 22227415      635.
11 Afghanistan Asia       2002    42.1 25268405      727.
12 Afghanistan Asia       2007    43.8 31889923      975.
13 Albania     Europe     1952    55.2  1282697     1601.
14 Albania     Europe     1957    59.3  1476505     1942.
15 Albania     Europe     1962    64.8  1728137     2313.
```

02 Data manipulation using base R

Extracting of samples and attributes

- extracting samples with the country name "Croatia" using a conditional expression.

```
> gapminder[gapminder$country == "Croatia", ]

# A tibble: 12 x 6
   country continent  year lifeExp     pop gdpPercap
   <fct>   <fct>     <int>   <dbl>   <int>     <dbl>
 1 Croatia Europe     1952    61.2 3882229     3119.
 2 Croatia Europe     1957    64.8 3991242     4338.
 3 Croatia Europe     1962    67.1 4076557     5478.
 4 Croatia Europe     1967    68.5 4174366     6960.
 5 Croatia Europe     1972    69.6 4225310     9164.
 6 Croatia Europe     1977    70.6 4318673    11305.
 7 Croatia Europe     1982    70.5 4413368    13222.
 8 Croatia Europe     1987    71.5 4484310    13823.
 9 Croatia Europe     1992    72.5 4494013     8448.
10 Croatia Europe     1997    73.7 4444595     9876.
11 Croatia Europe     2002    74.9 4481020    11628.
12 Croatia Europe     2007    75.7 4493312    14619.
```

02 Data manipulation using base R

Extracting of samples and attributes

- extracting samples with the country name "Croatia" using a conditional expression

  \+ Extracting population-attributes(pop) only

```
> gapminder[gapminder$country == "Croatia", "pop"]
# A tibble: 12 x 1
       pop
     <int>
 1 3882229
 2 3991242
 3 4076557
 4 4174366
 5 4225310
 6 4318673
 7 4413368
 8 4484310
 9 4494013
10 4444595
11 4481020
12 4493312
```

02 Data manipulation using base R

Extracting of samples and attributes

- extracting samples with the country name "Croatia" using a conditional expression

  \+ Extracting population-attributes(pop) only + life expectancy(lifeExp)

```
> gapminder[gapminder$country == "Croatia", c("lifeExp","pop")]
# A tibble: 12 x 2
   lifeExp     pop
     <dbl>   <int>
 1    61.2 3882229
 2    64.8 3991242
 3    67.1 4076557
 4    68.5 4174366
 5    69.6 4225310
 6    70.6 4318673
 7    70.5 4413368
 8    71.5 4484310
 9    72.5 4494013
10    73.7 4444595
11    74.9 4481020
12    75.7 4493312
```

If multiple attributes are to be extracted, enclose them as vectors using the c function.

02 Data manipulation using base R

Extracting of samples and attributes

- Extracting the life expectancy and population of Croatia since 1990

```
> gapminder[gapminder$country=="Croatia"&gapminder$year>1990,c("lifeExp","pop")]
# A tibble: 4 x 2
  lifeExp     pop
    <dbl>   <int>
1    72.5 4494013
2    73.7 4444595
3    74.9 4481020
4    75.7 4493312
```

Combining multiple conditional expressions into logical operators

02 Data manipulation using base R

Operation in row/column units

- Computing multiple items in a data frame using apply function provided by R

```
> apply(gapminder[gapminder$country=="Croatia", c("lifeExp","pop")], 2, mean)
     lifeExp          pop
7.005592e+01  4.289916e+06
```

02 Data manipulation using base R

- Data manipulation is done using various operators and functions provided by R around the data frame.
- Combine multiple conditional expressions into logical operators for more sophisticated extraction.
- In the process of exploring data, there are times when summary statistics of samples or quick operations in row/column units are required.
- Apply function provided by R enables multiple items that make up the data frame to be calculated at once.

03 Data manipulation using dplyr library

- Base R's data processing is based on **index-based data access**, but dplyr library is implemented as **a function of input-output relationships** such as filter or select, enabling users to use it more intuitively.
- Therefore, it is more efficient to use a specialized library for data manipulation.
- In the process of exploratory data analysis, visualization and data manipulation are very closely linked, requiring efficient manipulation techniques for visualization.

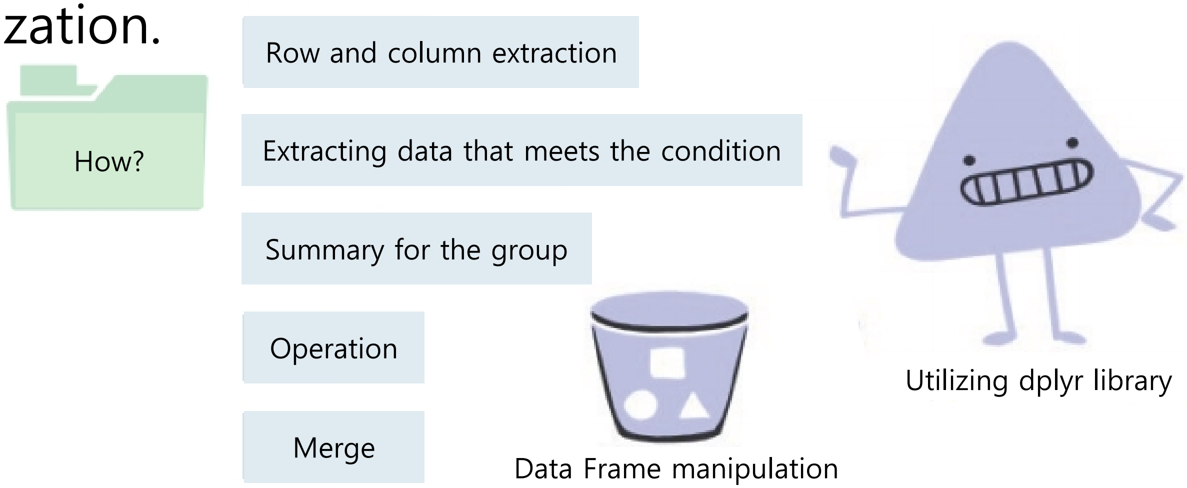


Figure 5-2 **dplyr library that is good for use in data frame processing**

03 Data manipulation using dplyr library

Extracting of samples and attributes

- Using select function
- Column names can be used without " " when specifying columns, so convenient.

```
> select(gapminder, country, year, lifeExp)
# A tibble: 1,704 x 3
   country      year lifeExp
   <fct>       <int>   <dbl>
 1 Afghanistan  1952    28.8
 2 Afghanistan  1957    30.3
 3 Afghanistan  1962    32.0
 4 Afghanistan  1967    34.0
 5 Afghanistan  1972    36.1
 6 Afghanistan  1977    38.4
 7 Afghanistan  1982    39.9
 8 Afghanistan  1987    40.8
 9 Afghanistan  1992    41.7
10 Afghanistan  1997    41.8
# ... with 1,694 more rows
```

03 Data manipulation using dplyr library

Extracting of samples and attributes

- Using filter function when extracting specific samples (row)
- Conditional expression configuration is similar to base R, but the command is concise because you do not need to enter the name of the data frame every time for indexing within the function.

```
> filter(gapminder, country=="Croatia")
# A tibble: 12 x 6
   country continent  year lifeExp     pop gdpPercap
   <fct>   <fct>     <int>   <dbl>   <int>     <dbl>
 1 Croatia Europe     1952    61.2 3882229     3119.
 2 Croatia Europe     1957    64.8 3991242     4338.
 3 Croatia Europe     1962    67.1 4076557     5478.
 4 Croatia Europe     1967    68.5 4174366     6960.
 5 Croatia Europe     1972    69.6 4225310     9164.
 6 Croatia Europe     1977    70.6 4318673    11305.
 7 Croatia Europe     1982    70.5 4413368    13222.
 8 Croatia Europe     1987    71.5 4484310    13823.
 9 Croatia Europe     1992    72.5 4494013     8448.
10 Croatia Europe     1997    73.7 4444595     9876.
11 Croatia Europe     2002    74.9 4481020    11628.
12 Croatia Europe     2007    75.7 4493312    14619.
```

03 Data manipulation using dplyr library

- Operation in row/column units
  - Using the group_by function, you can use factor-type attributes contained in the data frame to group the entire data.
  - Typically, summarise function is used one after another to calculate statistical indicators for each group at a time.

```
> summarize(gapminder, pop_avg = mean(pop))
# A tibble: 1 x 1
    pop_avg
      <dbl>
1 29601212.

> summarize(group_by(gapminder, continent), pop_avg = mean(pop))
# A tibble: 5 x 2
  continent    pop_avg
  <fct>          <dbl>
1 Africa      9916003.
2 Americas   24504795.
3 Asia       77038722.
4 Europe     17169765.
5 Oceania     8874672.

> summarize(group_by(gapminder, continent, country), pop_avg = mean(pop))
# A tibble: 142 x 3
# Groups:   continent [5]
   continent country                    pop_avg
   <fct>     <fct>                        <dbl>
 1 Africa    Algeria                  19875406.
 2 Africa    Angola                    7309390.
 3 Africa    Benin                     4017497.
 4 Africa    Botswana                   971186.
 5 Africa    Burkina Faso              7548677.
 6 Africa    Burundi                   4651608.
 7 Africa    Cameroon                  9816648.
 8 Africa    Central African Republic  2560963
 9 Africa    Chad                      5329256.
10 Africa    Comoros                    361684.
# ... with 132 more rows
```

03 Data manipulation using dplyr library

Continuous processing using %>% operator

- Connecting a series of processing tasks using the %>% operator

```
> gapminder %>% group_by(continent, country) %>% summarize(pop_avg = mean(pop))
# A tibble: 142 x 3
# Groups:   continent [5]
   continent country             pop_avg
   <fct>     <fct>                 <dbl>
 1 Africa    Algeria           19875406.
 2 Africa    Angola             7309390.
 · · · skip ·
   · ·
10 Africa    Comoros             361684.
# ... with 132 more rows
```

Same

```
> summarize(group_by(gapminder, continent, country), pop_avg = mean(pop))
# A tibble: 142 x 3
# Groups:   continent [5]
   continent country                pop_avg
```

03 Data manipulation using dplyr library

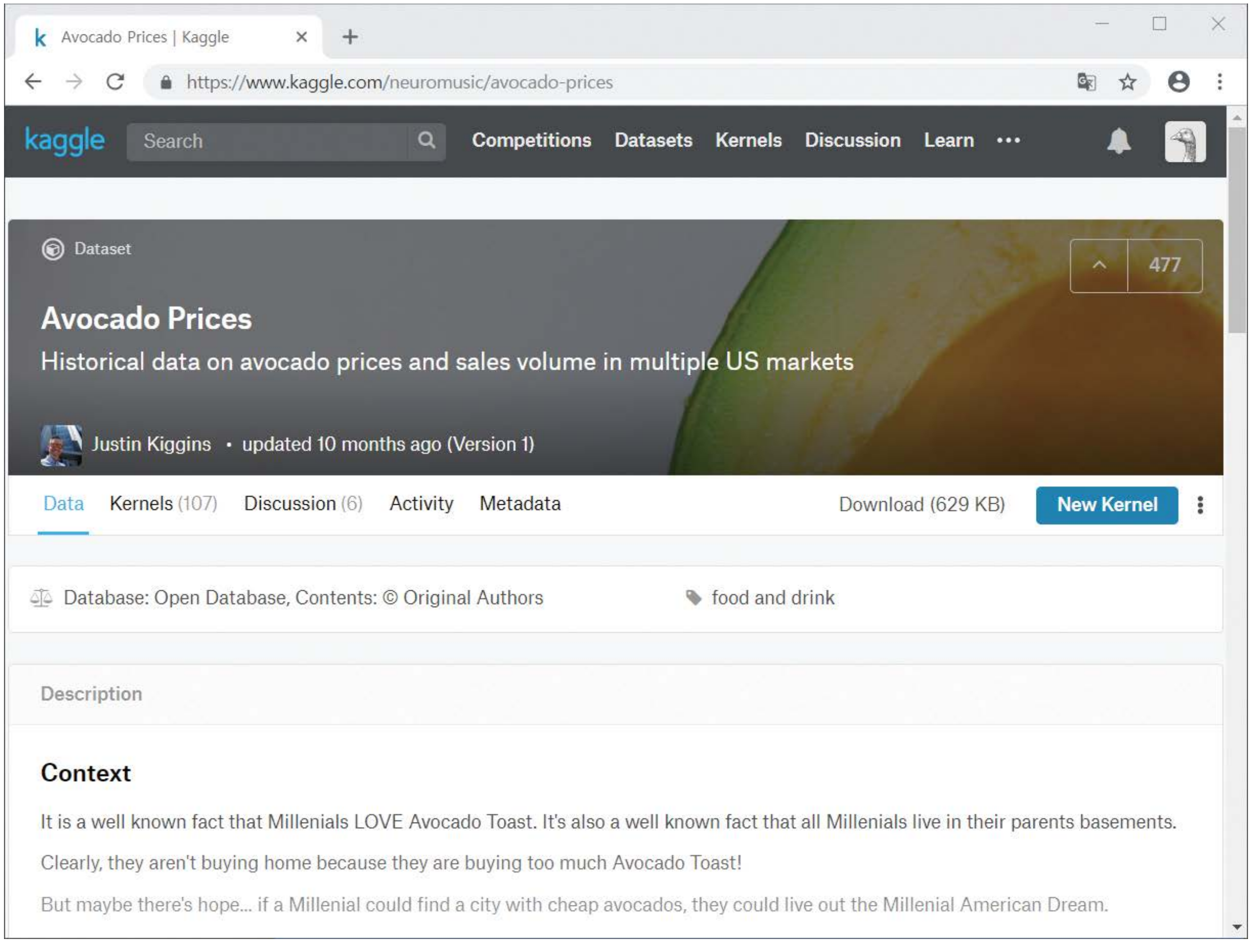
Continuous processing using %>% operator

- To process the results processed by one command with another, ❶ saves the intermediate result as a variable and processes it, but ❷ is connected using the %>% operator

```
> temp1=filter(gapminder, country=="Croatia")
> temp2=select(temp1, country, year, lifeExp)    ❶
> temp3=apply(temp2[ , c("lifeExp")], 2, mean)
> temp3
lifeExp
70.05592

> gapminder %>% filter(country=="Croatia") %>% select(country, year, lifeExp)
%>% summarize(lifeExp_avg = mean(lifeExp))    ❷
# A tibble: 1 x 1
  lifeExp_avg
        <dbl>
1        70.1
```

04 Actual data processing: Massive data summary

- Utilizing avocado data in Kaggle
  - https://www.kaggle.com/neuromusic/avocado-prices



04 Actual data processing: Massive data summary

- Exploring the configuration of the frame using the str function through reading the data set
  - There are 18,249 samples for a total of 14 attributes, it difficult to grasp the meaning at once.

```
> avocado <- read.csv("C:/Sources/avocado.csv", header = TRUE, sep = ",")
> str(avocado)
'data.frame':   18249 obs. of  14 variables:
 $ X           : int  0 1 2 3 4 5 6 7 8 9 ...
 $ Date        : Factor w/ 169 levels "2015-01-04","2015-01-11",..: 52 51 50 49 48 47 46 45 44 43 ...
 $ AveragePrice: num  1.33 1.35 0.93 1.08 1.28 1.26 0.99 0.98 1.02 1.07 ...
 $ Total.Volume: num  64237 54877 118220 78992 51040 ...
 $ X4046       : num  1037 674 795 1132 941 ...
 $ X4225       : num  54455 44639 109150 71976 43838 ...
 $ X4770       : num  48.2 58.3 130.5 72.6 75.8 ...
 $ Total.Bags  : num  8697 9506 8145 5811 6184 ...
 $ Small.Bags  : num  8604 9408 8042 5677 5986 ...
 $ Large.Bags  : num  93.2 97.5 103.1 133.8 197.7 ...
 $ XLarge.Bags : num  0 0 0 0 0 0 0 0 0 0 ...
 $ type        : Factor w/ 2 levels "conventional",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ year        : int  2015 2015 2015 2015 2015 2015 2015 2015 2015 2015 ...
 $ region      : Factor w/ 54 levels "Albany","Atlanta",..: 1 1 1 1 1 1 1 1 1 1 ...
```

04 Actual data processing: Massive data summary

Statistics in a group unit(1)

- Summary of total sales and average price attributes by region, respectively, to derive trends
- Using group_by and summarize functions in the dplyr library

```
> (x_avg = avocado %>% group_by(region) %>% summarize(V_avg = mean(Total.Volume),
P_avg = mean(AveragePrice)))
# A tibble: 54 x 3
   region                V_avg P_avg
   <fct>                 <dbl> <dbl>
 1 Albany               47538.  1.56
 2 Atlanta             262145.  1.34
 3 BaltimoreWashington 398562.  1.53
 4 Boise                42643.  1.35

 5 Boston              287793.  1.53
 6 BuffaloRochester     67936.  1.52
 7 California         3044324.  1.40
 8 Charlotte           105194.  1.61
 9 Chicago             395569.  1.56
10 CincinnatiDayton    131722.  1.21
# ... with 44 more rows
```

04 Actual data processing: Massive data summary

Statistics in a group unit(2)

- Re-segmentation of regional characteristics, by year.

```
> (x_avg = avocado %>% group_by(region, year) %>% summarize(V_avg = mean(Total.Volume),
P_avg = mean(AveragePrice)))
# A tibble: 216 x 4
# Groups:   region [54]
   region               year   V_avg P_avg
   <fct>               <int>   <dbl> <dbl>
 1 Albany               2015  38749.  1.54
 2 Albany               2016  50619.  1.53
 3 Albany               2017  49355.  1.64
 4 Albany               2018  64249.  1.44
 5 Atlanta              2015 223382.  1.38
 6 Atlanta              2016 272374.  1.21
 7 Atlanta              2017 271841.  1.43
 8 Atlanta              2018 342976.  1.29
 9 BaltimoreWashington  2015 390823.  1.37
10 BaltimoreWashington  2016 393210.  1.59
# ... with 206 more rows
```

04 Actual data processing: Massive data summary

Statistics in a group unit(3)

- Re-segmentation based on organic status(type) again.

```
> x_avg = avocado %>% group_by(region, year, type) %>% summarize(V_avg = mean(Total.
Volume), P_avg = mean(AveragePrice))
> x_avg
# A tibble: 432 x 5
# Groups:   region, year [216]
   region    year type           V_avg P_avg
   <fct>    <int> <fct>          <dbl> <dbl>
 1 Albany    2015 conventional  76209.  1.17

 2 Albany    2015 organic        1289.  1.91
 3 Albany    2016 conventional  99453.  1.35
 4 Albany    2016 organic        1784.  1.72
 5 Albany    2017 conventional  95779.  1.53
 6 Albany    2017 organic        2931.  1.75
 7 Albany    2018 conventional 124161.  1.34
 8 Albany    2018 organic        4338.  1.53
 9 Atlanta   2015 conventional 440346.  1.05
10 Atlanta   2015 organic        6417.  1.71
# ... with 422 more rows
```

04 Actual data processing: Massive data summary

Statistics in a group unit(4)

- Get summary statistics of total sales and average prices based on region, year and whether organic cultivation from vast sample data.
- The most effective way to observe these statistics by year is visualization we will learn in chapter 6.
- In this chapter, only refer to the visualized results.

04 Actual data processing: Massive data summary

Statistics in a group unit(5)

```
> library(ggplot2)
> x_avg %>% filter(region != "TotalUS") %>% ggplot(aes(year, V_avg, col = type)) +
geom_line() + facet_wrap(~region)
```

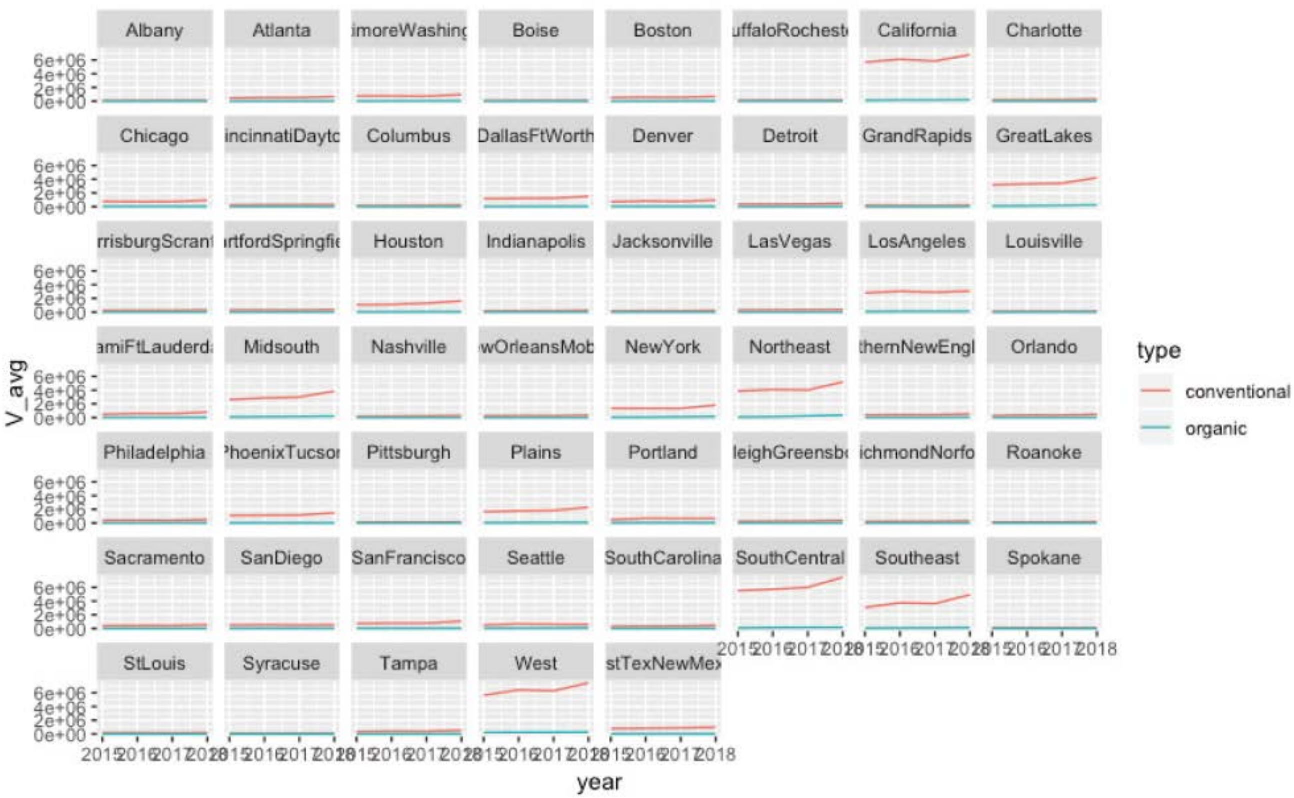


Figure 5-4 Graphs that visualize changes in total annual sales of avocados

04 Actual data processing: Massive data summary

Data sorting and searching(1)

- Sorting and searching allow you to observe data in detail.
- By using arrange function to sort the data by the average price of the total sales volume, you can find the year and region of the peak as well as the ranking of sales.

```
> arrange(x_avg, desc(V_avg))
# A tibble: 432 x 5
# Groups:   region, year [216]
   region        year type             V_avg P_avg
   <fct>        <int> <fct>            <dbl> <dbl>
 1 TotalUS       2018 conventional 42125533. 1.06
 2 TotalUS       2016 conventional 34043450. 1.05
 3 TotalUS       2017 conventional 33995658. 1.22
 4 TotalUS       2015 conventional 31224729. 1.01
 5 SouthCentral  2018 conventional  7465557. 0.806
 6 West          2018 conventional  7451445. 0.981
 7 California    2018 conventional  6786962. 1.08
 8 West          2016 conventional  6404892. 0.916
 9 West          2017 conventional  6279482. 1.10
10 California    2016 conventional  6105539. 1.05
# ... with 422 more rows
```

04 Actual data processing: Massive data summary

Data sorting and searching(2)

- Data sets often contain intermediate statistical values, so care needs to be taken
- To search for the maximum value may be used max function, but it is safer to use arrange function to check the property value.

```
> x_avg1 = x_avg %>% filter(region != "TotalUS")

> After excluding TotalUS, we can process by using statistical function directly
> x_avg1[x_avg$V_avg == max(x_avg1$V_avg), ]
# A tibble: 1 x 5
# Groups:   region, year [1]
  region        year type              V_avg P_avg
  <fct>        <int> <fct>             <dbl> <dbl>
1 SouthCentral  2018 conventional 7465557. 0.806
```

04 Actual data processing: Massive data summary

Utilizing date-type data(1)

- The date-type attribute consists of 31 days for one month and 12 months for one year, so special processing is required because the gap between data is not constant and can be applied incorrectly in visual or modeling phases.
- Each attribute contains three attributes (annual-month-day) information, so you can analyze the data more carefully when properly processed.

```
> x_avg1 = x_avg %>% filter(region != "TotalUS")

> After excluding TotalUS, we can process by using statistical function directly
> x_avg1[x_avg$V_avg == max(x_avg1$V_avg), ]
# A tibble: 1 x 5
# Groups:   region, year [1]
  region        year type            V_avg P_avg
  <fct>        <int> <fct>           <dbl> <dbl>
1 SouthCentral  2018 conventional 7465557. 0.806
```

04 Actual data processing: Massive data summary

Utilizing date-type data(2)

- Summary of avocado sales information to monthly average instead of the yearly average
- Using month function provided by lubridate library to extract month from the date attribute date type
- You can also use year or day functions.

```
> library(lubridate)
> (x_avg = avocado %>% group_by(region, year, month(Date), type) %>% summarize(V_
avg = mean(Total.Volume), P_avg = mean(AveragePrice)))
# A tibble: 4,212 x 6
# Groups:   region, year, month(Date) [, 106]
   region  year `month(Date)` type          V_avg P_avg
   <fct>  <int>         <dbl> <fct>         <dbl> <dbl>
 1 Albany  2015             1 conventional 42932.  1.17
 2 Albany  2015             1 organic       1198.  1.84
 3 Albany  2015             2 conventional 52343.  1.03
 4 Albany  2015             2 organic       1334.  1.76
 5 Albany  2015             3 conventional 50659.  1.06
 6 Albany  2015             3 organic       1444.  1.83
 7 Albany  2015             4 conventional 48594.  1.17
 8 Albany  2015             4 organic       1402.  1.89
 9 Albany  2015             5 conventional 97216.  1.26
10 Albany  2015             5 organic       1836.  1.94
# ... with 4,202 more rows
```

04 Actual Data Processing : Processing for modeling

- UCI Repository is a repository of data designed for experimental analysis of machine learning algorithms
- Download wine.data.txt to C:/Sources in
  https://archive.ics.uci.edu/ml/datasets/Wine
- One observation value consists of 14 numerical data. The remaining 13 numerical data, excluding the first row to indicate the type of wine, are based on an analysis of the chemical composition of the wine. It is widely known as a representative example of modeling the interrelationships between the types of wine and its component analyses.
- Measurement properties within the data frame, that is, column names, were not recorded as headers.
- This information is described in a separate file and should be incorporated within the data frame by the user or used separately from numerical data.

04 Actual Data Processing : Processing for modeling

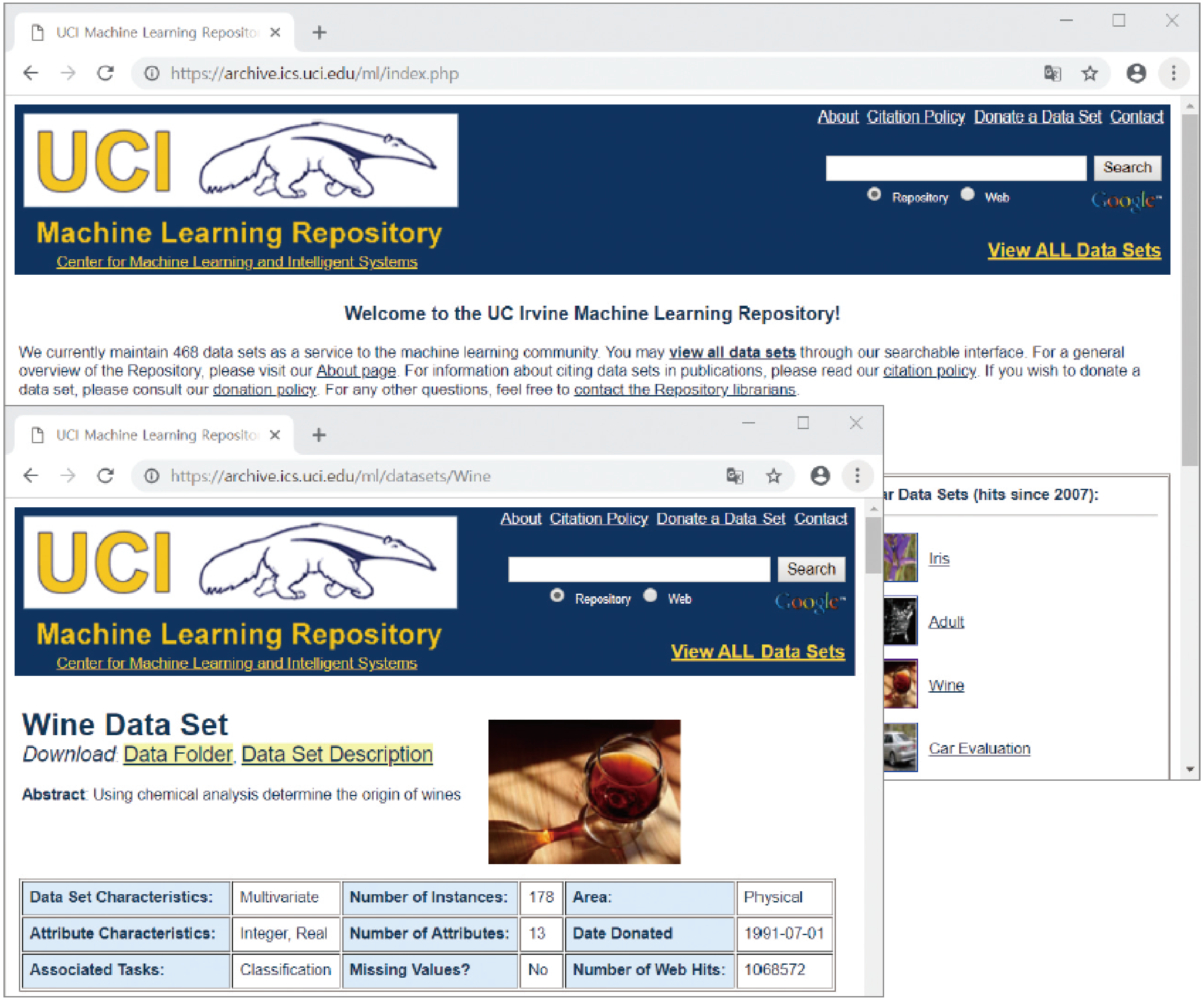


Figure 5-5 UCI Machine Learning Repository

04 Actual Data Processing : Processing for modeling

Read and write column names for data frames(1)

- Let's save the following content in a separate file called wine.name.txt

```
1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of ash
5) Magnesium
6) Total phenols
7) Flavanoids
8) Alcohol
9) Malic acid
10) Ash
11) Alcalinity of ash
12) Magnesium
13) Total phenols
```

04 Actual Data Processing : Processing for modeling

Read and write column names for data frames(2)

- Read wine.name.txt file and specify it as the column name of the wine data.
- Use substr function to extract part of a string.
- After attribute assignment, wine data can be explored and modeled more effectively.

```
> wine = read.table("C:/Sources/wine.data.txt", header = FALSE, sep = ",")
> head(wine)
  X1 X14.23 X1.71 X2.43 X15.6 X127 X2.8 X3.06 X.28 X2.29 X5.64 X1.04 X3.92 X1065
1  1  13.20  1.78  2.14  11.2  100 2.65  2.76 0.26  1.28  4.38  1.05  3.40  1050
2  1  13.16  2.36  2.67  18.6  101 2.80  3.24 0.30  2.81  5.68  1.03  3.17  1185
3  1  14.37  1.95  2.50  16.8  113 3.85  3.49 0.24  2.18  7.80  0.86  3.45  1480
4  1  13.24  2.59  2.87  21.0  118 2.80  2.69 0.39  1.82  4.32  1.04  2.93   735
5  1  14.20  1.76  2.45  15.2  112 3.27  3.39 0.34  1.97  6.75  1.05  2.85  1450
6  1  14.39  1.87  2.45  14.6   96 2.50  2.52 0.30  1.98  5.25  1.02  3.58  1290

> n = readLines("C:/wine.name.txt")
> n
[1] "1) Alcohol"           "2) Malic acid"        "3) Ash"               "4)
Alcalinity of ash" "5) Magnesium"         "6) Total phenols"
[7] "7) Flavanoids"

> names(wine)[2:14] <- substr(n, 4, nchar(n))
> names(wine)
 [1] "X1"                "Alcohol"           "Malic acid"        "Ash"
"Alcalinity of ash" "Magnesium"         "Total phenols"
 [8] "Flavanoids"        "Alcohol"           "Malic acid"        "Ash"
"Alcalinity of ash" "Magnesium"         "Total phenols"
```

04 Actual Data Processing : Processing for modeling

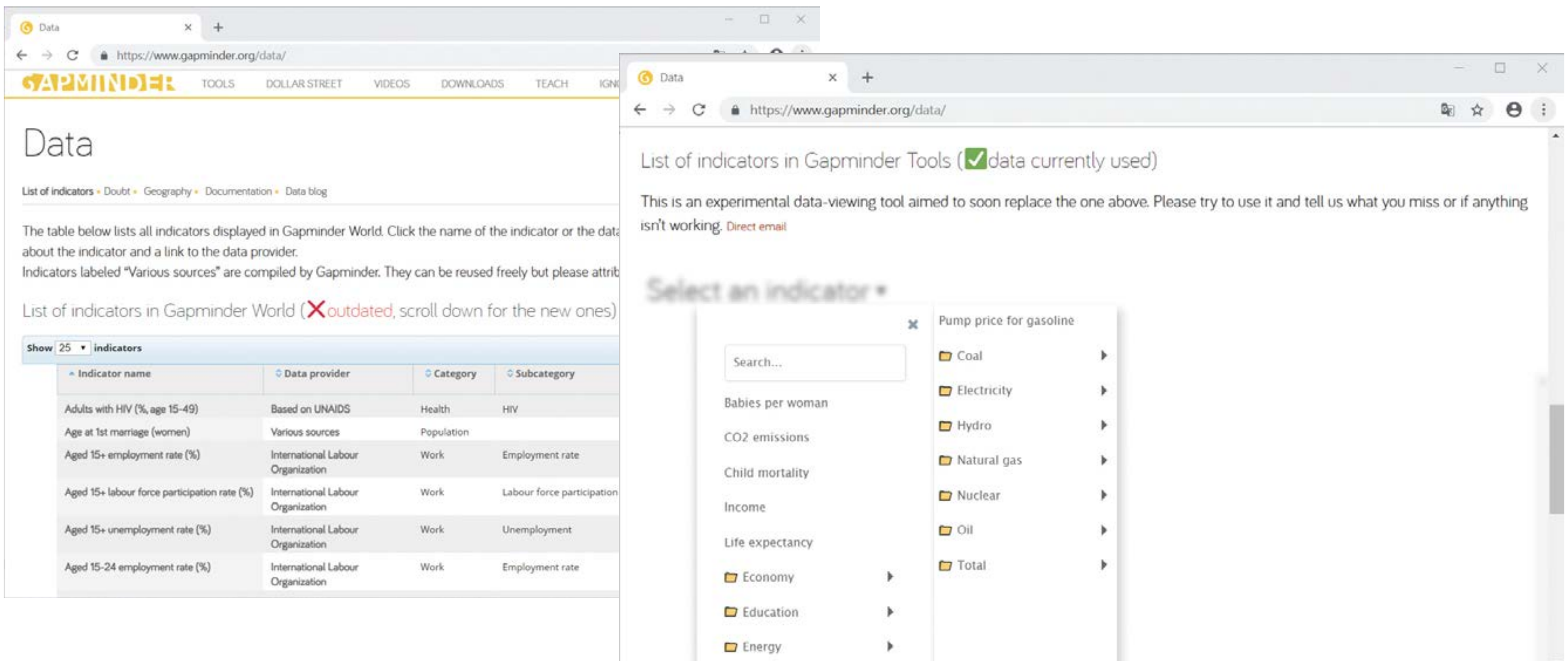
Splitting data sets

- Learning data required to learn modeling, and test data to verify that the models obtained are appropriate, are obtained by dividing a given set of data by a certain percentage.
- It is important to take random samples and divide them.
- Simple to use the sample_frac or sample_n function provided by dplyr.

```
> train_set = sample_frac(wine, 0.6)
> str(train_set)
'data.frame'   : 107 obs. of  14 variables:
 $ id          : Factor w/ 3 levels "1","2","3": 3 3 1 2 1 3 2 1 3 3 ...
 $ Alcohol     : num  13.2 13.5 13.6 11.8 13.9 ...
 $ Malic acid  : num  3.3 3.17 1.81 1.72 1.73 4.61 2.08 1.81 3.9 3.88 ...
 $ Ash         : num  2.28 2.72 2.7 1.88 2.27 2.48 1.7 2.61 2.36 2.2 ...
 …

> test_set = setdiff(wine, train_set)
> str(test_set)
'data.frame'   : 71 obs. of  14 variables:
 $ id          : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
 $ Alcohol     : num  14.2 14.4 14.4 14.3 13.8 ...
 $ Malic acid  : num  1.71 1.95 1.87 1.92 1.57 3.8 2.05 1.77 1.83 1.81 ...
 $ Ash         : num  2.43 2.5 2.45 2.72 2.62 2.65 3.22 2.62 2.36 2.41 ...
```

04 Actual Data Processing: Change data structure

- The data in the Gapminder package is only a fraction of the data provided by the Gapminder website.
- The overall analysis of multiple indicators requires processing to organize observations of various items and incorporate them into a single data frame.



04 Actual Data Processing: Change data structure

- Download each power production and usage data from the Gapminder website and process it into a single data frame.
- Using the search function provided on the site, you can acquire data that records electricity production per person and electricity consumption per person.

04 Actual Data Processing: Change data structure

- Download and open the electricity production per man data file (electricity_generation_per_per_person.csv).
  - For a total of 65 countries, per capita electricity output was recorded for 33 years (1985 to 2016)
- The character x was unexpectedly added before the year value.
  - Occasionally this happens due to text encoding problems

```
> elec_gen = read.csv("C:/Sources/electricity_generation_per_person.csv", header =
TRUE, sep = ",")
> names(elec_gen)
 [1] "country" "X1985"   "X1986"   "X1987"   "X1988"   "X1989"   "X1990"   "X1991"   "X1992"   "X1993"   "X1994"   "X1995"
[13] "X1996"   "X1997"   "X1998"   "X1999"   "X2000"   "X2001"   "X2002"   "X2003"   "X2004"   "X2005"   "X2006"   "X2007"
[25] "X2008"   "X2009"   "X2010"   "X2011"   "X2012"   "X2013"   "X2014"   "X2015"   "X2016"

> names(elec_gen) = substr(names(elec_gen), 2, nchar(names(elec_gen)))
> names(elec_gen)
 [1] "ountry" "1985"   "1986"   "1987"   "1988"   "1989"   "1990"   "1991"   "1992"   "1993"   "1994"   "1995"   "1996"
[14] "1997"   "1998"   "1999"   "2000"   "2001"   "2002"   "2003"   "2004"   "2005"   "2006"   "2007"   "2008"   "2009"
[27] "2010"   "2011"   "2012"   "2013"   "2014"   "2015"   "2016"
```

04 Actual Data Processing: Change data structure

- Use the names and substr function we learned earlier to organize neatly.

```
> elec_gen = read.csv("C:/Sources/electricity_generation_per_person.csv", header =
TRUE, sep = ",")
> names(elec_gen)
 [1] "country" "X1985"   "X1986"   "X1987"   "X1988"   "X1989"   "X1990"   "X1991"   "X1992"   "X1993"   "X1994"   "X1995"
[13] "X1996"   "X1997"   "X1998"   "X1999"   "X2000"   "X2001"   "X2002"   "X2003"   "X2004"   "X2005"   "X2006"   "X2007"
[25] "X2008"   "X2009"   "X2010"   "X2011"   "X2012"   "X2013"   "X2014"   "X2015"   "X2016"

> names(elec_gen) = substr(names(elec_gen), 2, nchar(names(elec_gen)))
> names(elec_gen)
 [1] "ountry" "1985"   "1986"   "1987"   "1988"   "1989"   "1990"   "1991"   "1992"   "1993"   "1994"   "1995"   "1996"
[14] "1997"   "1998"   "1999"   "2000"   "2001"   "2002"   "2003"   "2004"   "2005"   "2006"   "2007"   "2008"   "2009"
[27] "2010"   "2011"   "2012"   "2013"   "2014"   "2015"   "2016"
```

04 Actual Data Processing: Change data structure

- Download electrical usage data (electricity_use__per_person.csv) in the same way
- Removing unnecessary characters from year names
- For a total of 138 countries, per capita electricity usage was recorded for 56 years (1960-2014)

```
> elec_use = read.csv("C:/Sources/electricity_use_per_person.csv", header = TRUE,
sep = ",")
> names(elec_use)[2:56] = substr(names(elec_use)[2:56], 2, nchar(names(elec_use)
[2:56]))
```

country	1985	1986	1987	1988	1989	1990	1991
Algeria	544.0	559.0	532.0	568.0	607.0	621.0	653.0
Argentina	1490.0	1590.0	1660.0	1670.0	1580.0	1560.0	1620.0
Australia	7860.0	8100.0	8360.0	8670.0	9020.0	9130.0	9150.0
Austria	5850.0	5860.0	6610.0	6400.0	6530.0	6530.0	6620.0
Azerbaijan	3110.0	3180.0	3320.0	3360.0	3270.0	3200.0	3170.0
Bangladesh	48.6	50.1	56.8	64.8	68.7	72.8	76.1
Belarus	3330.0	3620.0	3710.0	3760.0	3780.0	3870.0	3790.0
Belgium	5780.0	5910.0	6370.0	6560.0	6770.0	7090.0	7170.0
Brazil	1430.0	1460.0	1440.0	1490.0	1510.0	1490.0	1540.0
Bulgaria	4640.0	4660.0	4850.0	5040.0	4980.0	4770.0	4660.0

(a) Per capita electricity output table by View commands

country	1960	1961	1962	1963	1964	1965	1966
Albania	NA	NA	NA	NA	NA	NA	NA
Algeria	NA	NA	NA	NA	NA	NA	NA
Angola	NA	NA	NA	NA	NA	NA	NA
Argentina	NA	NA	NA	NA	NA	NA	NA
Armenia	NA	NA	NA	NA	NA	NA	NA
Australia	1830.0	1950.0	2010	2210	2420	2630	2770
Austria	1810.0	1880.0	2010	2120	2230	2310	2380
Azerbaijan	NA	NA	NA	NA	NA	NA	NA
Bahrain	NA	NA	NA	NA	NA	NA	NA
Bangladesh	NA	NA	NA	NA	NA	NA	NA

(b) Per capita electricity usage table by View commands

Figure 5-7 Per capita electricity production and electricity usage data downloaded

04 Actual Data Processing: Change data structure

Merge two data frames(1)

- Composition of common data frames you've seen so far
  - Placing one property in one column,
  - and one row in the same configuration as the one recorded sample.
  - Each measurement year value is named as a column, recording years of data in a row
- The country name, year, electricity production, and usage need to be adjusted to correspond column each.
- Using gather function

04 Actual Data Processing: Change data structure

Merge two data frames(2)

- If you assign the year to be the delimiter in the newly created data frame to the key and the property name of the measurement to value, gather function will reconstruct the data frame as follows

```
> library(tidyr)
> elec_gen_df = gather(elec_gen, -country,  key = "year", value = "ElectricityGeneration")
> elec_use_df = gather(elec_use, -country,  key = "year", value = "ElectricityUse")
```

elec_gen_df

Filter

	country	year	ElectricityGeneration
1	Algeria	1985	544.0
2	Argentina	1985	1490.0
3	Australia	1985	7860.0
4	Austria	1985	5850.0
5	Azerbaijan	1985	3110.0
6	Bangladesh	1985	48.6
7	Belarus	1985	3330.0
8	Belgium	1985	5780.0
9	Brazil	1985	1430.0
10	Bulgaria	1985	4640.0
11	Canada	1985	17800.0

Showing 1 to 11 of 2,080 entries

elec_use_df

Filter

column 1: factor with 1 levels

	country	year	ElectricityUse
1	Australia	1960	1830.0
2	Austria	1960	1810.0
3	Belgium	1960	1580.0
4	Canada	1960	5630.0
5	Denmark	1960	1090.0
6	Finland	1960	1870.0
7	France	1960	1460.0
8	Germany	1960	1590.0
9	Greece	1960	242.0
10	Iceland	1960	2610.0
11	Ireland	1960	695.0

Showing 1 to 11 of 1,375 entries

Figure 5-8 Electricity production per man and electricity usage, converted by gather function(before merging)

04 Actual Data Processing: Change data structure

Merge two data frames(3)

- Merge reconstructed data frames into one data frame using merge function

```
> elec_gen_use = merge(elec_gen_df, elec_use_df)
```

elec_gen_use

Filter

	country	year	ElectricityGeneration	ElectricityUse
1	Australia	1985	7860	7010
2	Australia	1986	8100	7310
3	Australia	1987	8360	7530
4	Australia	1988	8670	7800
5	Australia	1989	9020	8130
6	Australia	1990	9130	8530
7	Australia	1991	9150	8520
8	Australia	1992	9230	8550
9	Australia	1993	9350	8700
10	Australia	1994	9510	8810
11	Australia	1995	9690	8990

Showing 1 to 11 of 690 entries

Figure 5-9 Electricity production per man and electricity usage (after merging)

Significance of data processing

- In addition to extracting data areas, it includes several tasks for intuitive and easy retrieval of data.
- Older than any other analytical technique, the basic technology of data science. You can feel a sense of accomplishment by continuously processing large and complex blocks of data.

Figure 5-10 **The meaning of data processing**

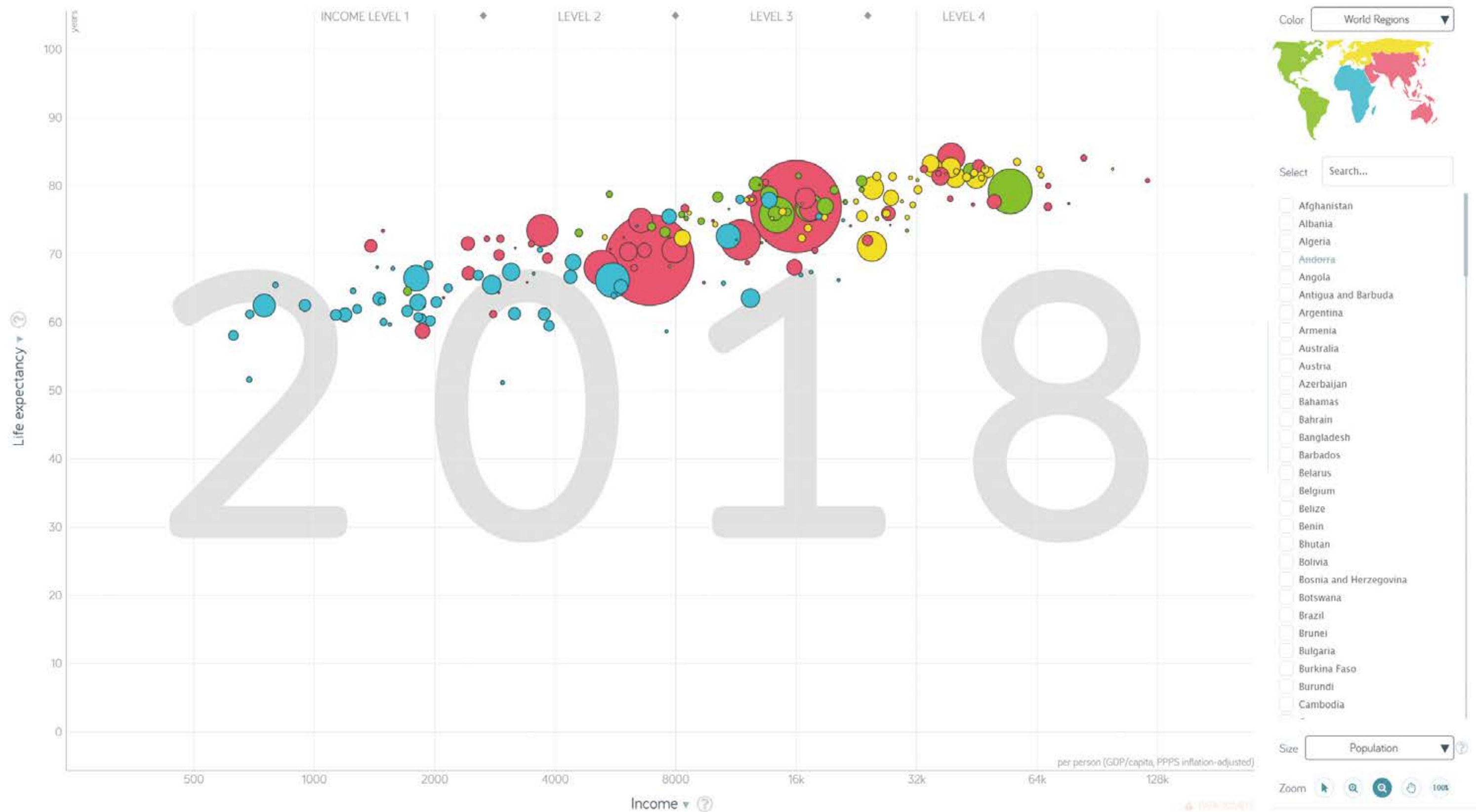
Significance of data processing

- Data transformation is that change the way observers think and view of the data
- Data processing is by no means a mechanical or simple task, cutting out unnecessary parts so that the meaning of the data is well expressed, and requiring constant thinking to give credibility and consistency.
- **<u>Understanding and processing data should proceed together.</u>**

Figure 5-11 **Data Processing = Changing the perspective of data observation**

PREVIEW

- Data consists of numerous attributes and samples, it is difficult to see what it means at a glance.
- The best way to gain insight and communicate the meaning of your data is to visualize it.



Bubble chart with the visual representation of gapminder data(https://www.gapminder.org)

01 What is Data Visualization? : The Need for Data Visualization

- Visualization of data is not an option in the process of observing the data, but an essential process that must be taken.

anscombe

	x1	x2	x3	x4	y1	y2	y3	y4
1	10	10	10	8	8.04	9.14	7.46	6.58
2	8	8	8	8	6.95	8.14	6.77	5.76
3	13	13	13	8	7.58	8.74	12.74	7.71
4	9	9	9	8	8.81	8.77	7.11	8.84
5	11	11	11	8	8.33	9.26	7.81	8.47
6	14	14	14	8	9.96	8.10	8.84	7.04
7	6	6	6	8	7.24	6.13	6.08	5.25
8	4	4	4	19	4.26	3.10	5.39	12.50
9	12	12	12	8	10.84	9.13	8.15	5.56
10	7	7	7	8	4.82	7.26	6.42	7.91
11	5	5	5	8	5.68	4.74	5.73	6.89

Figure 6-1 Anscombe's four data sets

Four data sets

data1=(x1, y1)
data2=(x2, y2)
data3=(x3, y3)
data4=(x4, y4)

01 What is Data Visualization? : The Need for Data Visualization

- Each of the 11 data samples comprising data1~data4 has the same mean, variance, and correlation between x and y

```
> # mean
> apply(anscombe, 2, mean)
      x1       x2       x3       x4       y1       y2       y3       y4 
9.000000 9.000000 9.000000 9.000000 7.500909 7.500909 7.500000 7.500909 

> # variance
> apply(anscombe, 2, var)
       x1        x2        x3        x4        y1        y2        y3        y4 
11.000000 11.000000 11.000000 11.000000  4.127269  4.127629  4.122620  4.123249 

> # correlation (correlation coefficient)
> cor(anscombe$x1, anscombe$y1)
[1] 0.8164205

> cor(anscombe$x2, anscombe$y2)
[1] 0.8162365

> cor(anscombe$x3, anscombe$y3)
[1] 0.8162867

> cor(anscombe$x4, anscombe$y4)
[1] 0.8165214
```

01 What is Data Visualization? : The Need for Data Visualization

- Linear regression is also almost identical

$$y1 = 0.5001 \times x1 + 3.0001$$
$$y2 = 0.500 \times x2 + 3.001$$
$$y3 = 0.4997 \times x3 + 3.0025$$
$$y4 = 0.4999 \times x4 + 3.0017$$

- Comparing statistical indicators with analytical figures alone, four sets of data can be determined to be almost identical

- However, if you graph it, it is data with different distributions.

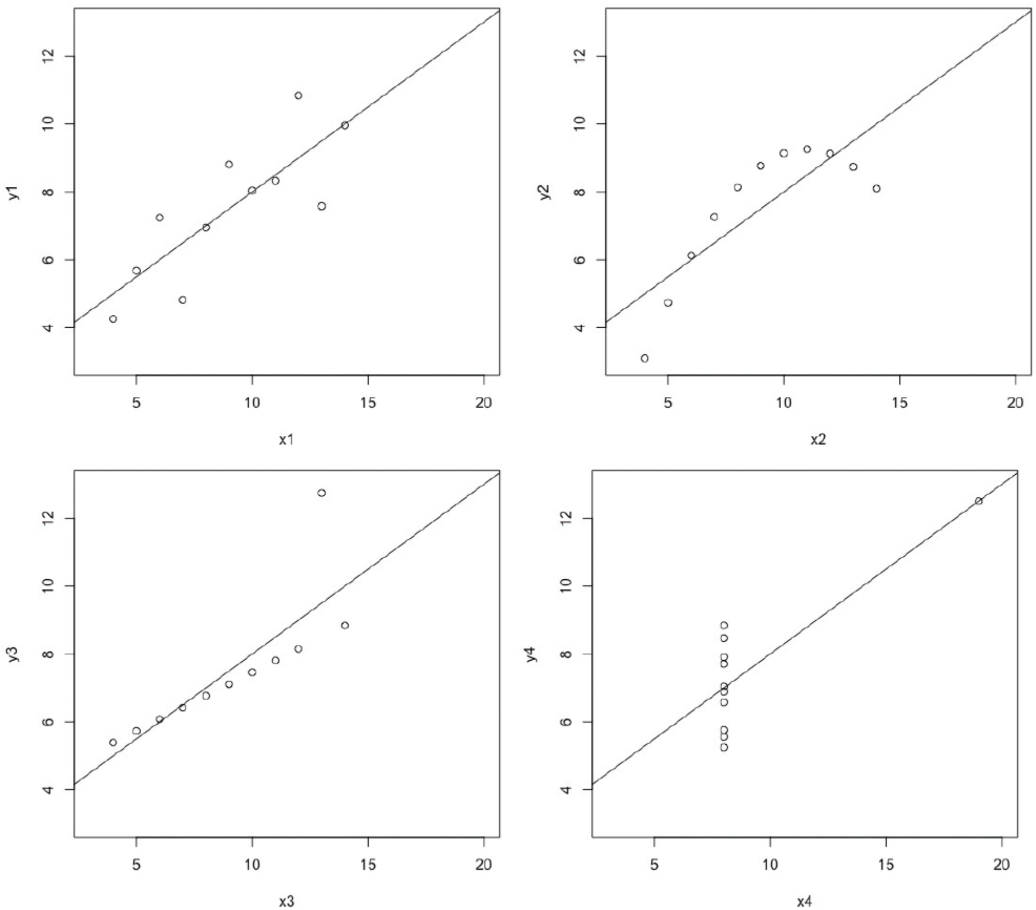


Figure 6-2 Graphs that visualize the distribution of four data sets (the line is linear regression)

01 What is Data Visualization? : Basic elements of visualization

- gapminder data contains population data from 1952-2007 for a total of 142 countries(5 continents) at 5-year intervals.
- It is convenient to observe trends in population change by continent rather than by individual countries

```
> library(gapminder)
> library(dplyr)
> y <- gapminder %>% group_by(year, continent) %>% summarize(c_pop = sum(pop))
> head(y, 20)
# A tibble: 20 x 3
# Groups:   year [4]

    year continent      c_pop
   <int> <fct>          <dbl>
 1  1952 Africa     237640501
 2  1952 Americas   345152446
 3  1952 Asia      1395357351
 4  1952 Europe     418120846
 5  1952 Oceania     10686006
 6  1957 Africa     264837738
 7  1957 Americas   386953916
 8  1957 Asia      1562780599
 9  1957 Europe     437890351
10  1957 Oceania     11941976
11  1962 Africa     296516865
```

01 What is Data Visualization? : Basic elements of visualization

- Visualizing the results summarized in numbers using plot function provided by Base R

```
> plot(y$year, y$c_pop)
```

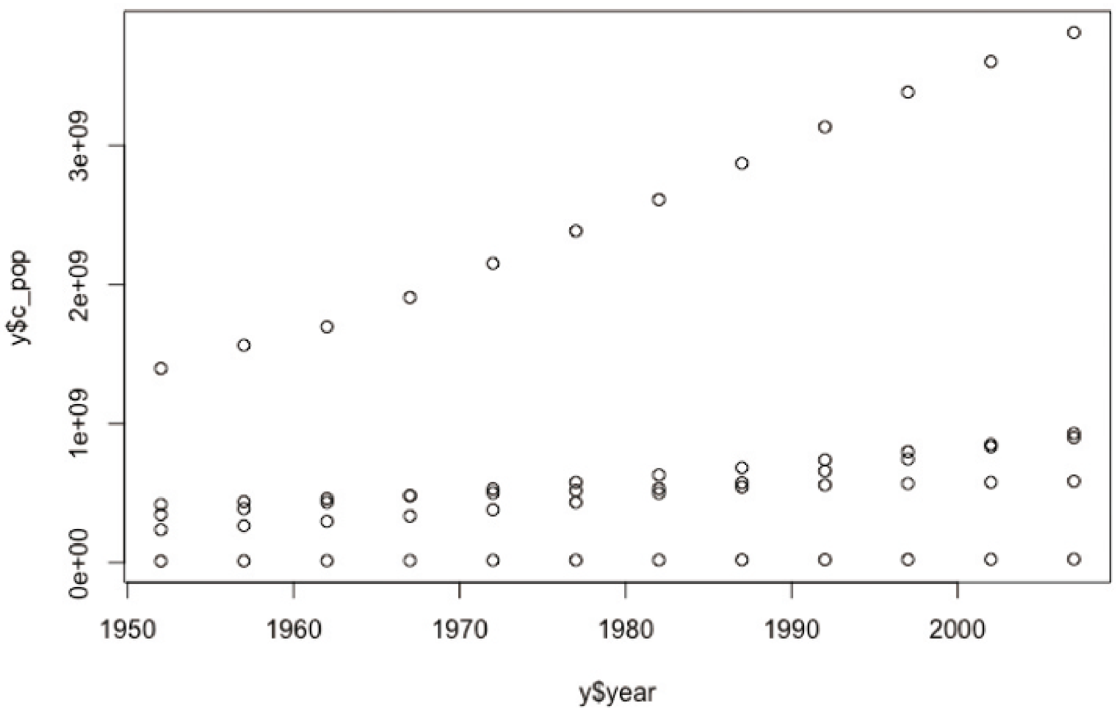


Figure 6-3 Basic visualization graph using the plot function

01 What is Data Visualization? : Basic elements of visualization

- Since data from multiple continents are displayed on a single graph, we have to use additional options to specify different colors or shapes of markers.

```
> plot(y$year, y$c_pop, col = y$continent)
```

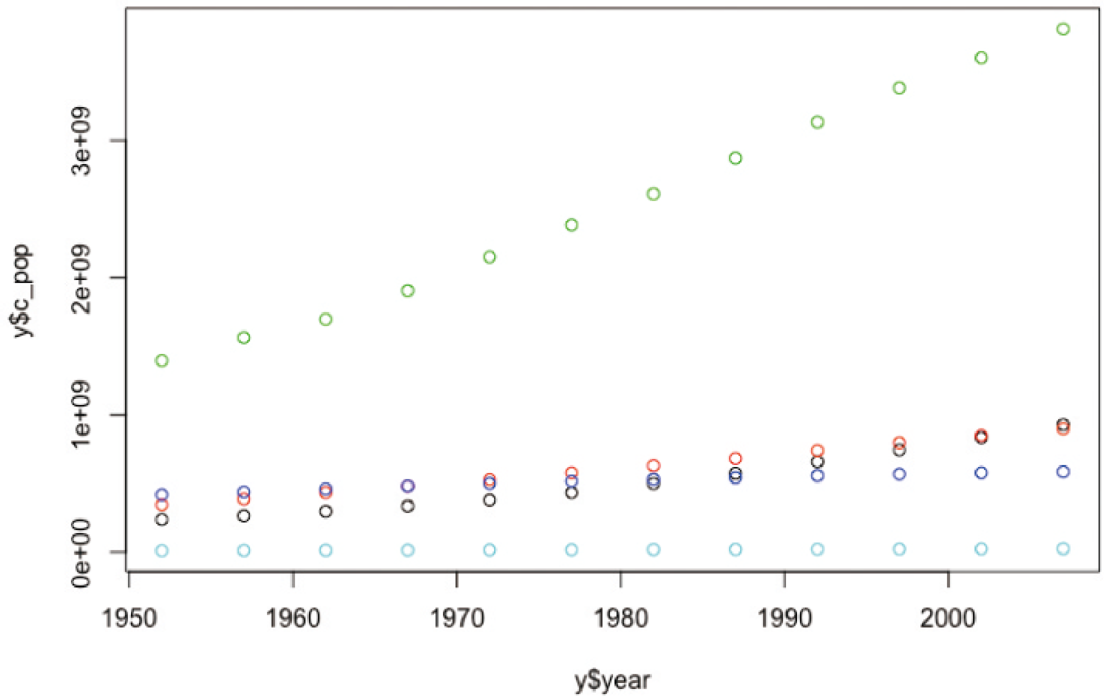


Figure 6-4 Adding color marker options to graphs in [Figure 6-3]

01 What is Data Visualization? : Basic elements of visualization

- You can also enter (❶) the number of markers you need directly, but to express the meaning of the command well, enter using the variable you are print (❷).

```
❶ > plot(y$year, y$c_pop, col = y$continent, pch = c(1:5))
❷ > plot(y$year, y$c_pop,col = y$continent, pch = c(1:length(levels(y$continent))))
```

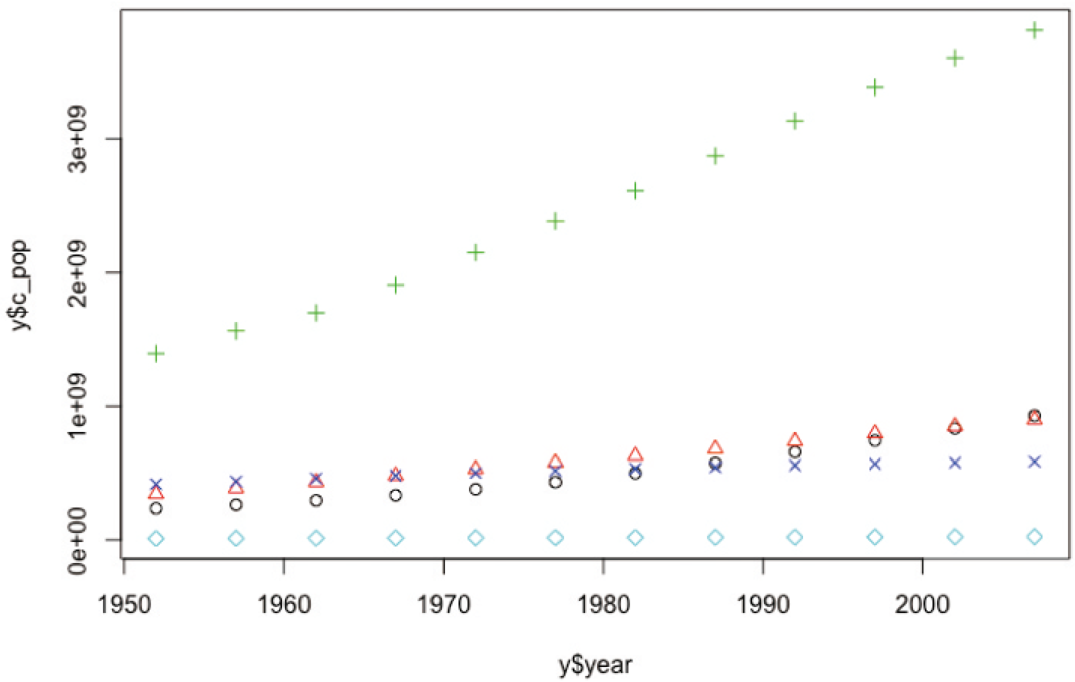
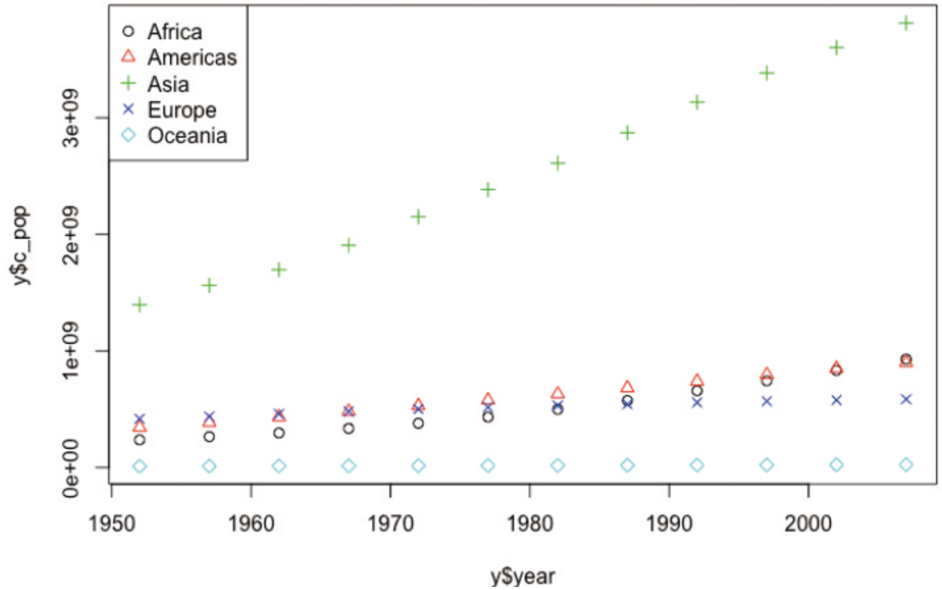


Figure 6-5 Adding shape marker options to graphs in [Figure 6-4]

01 What is Data Visualization? : Basic elements of visualization

- Displaying a legend describing each marker in the blank space of the plot completes the basic visualization.

```
> # Specifies the number of legends using a number
> legend("topleft", legend = 5, pch = c(1:5), col = c(1:5))

> # Specifies the number of legends to match the number of data
> legend("topleft", legend = levels((y$continent)), pch = c(1:length(levels(y$continent))), col = c(1:length(levels(y$continent))))
```



By visually identifying trends in which the population of the Asian continent is growing particularly rapidly, we can also guess some future trends. In other words, <u>visualized results also being a role in inducing intuitive predictions</u>.

Figure 6-6 Adding a legend to the graph in [Figure 6-5]

02 Basic function of visualization: effective observation of large amounts of data

- Visualization serves to enable data to be interpreted correctly, while also allowing large amounts of data to be observed effectively.
- In the recent data science field, complexity has increased as more and more data is being handled to increase reliability.
- The effects of visualization
    - Intuitive Insight can be obtained.
    - We can clearly understand the core.
    - In addition to the average trend, we can also find outliers.
    - We can quickly find problems in data.

02 Basic function of visualization: effective observation of large amounts of data

Intuitive understanding of gapminder data (1)

- Using data frame summary functions such as glimpse and str to gain some insight into the size and nature of your data.
- If visualizations are available, data can be intuitively understood without the process of extracting summary statistics.

Intuitive understanding of gapminder data (2)

- Using the attributes of the original data as much as possible to display all the samples on the graph. However, we can also check the range and characteristics of gdpPercap, lifeExp, pop items, relative differences, and approximate correlation using markers that are distinguished by continent or country.

```
> plot(gapminder$gdpPercap, gapminder$lifeExp, col=gapminder$continent)
> legend("bottomright", legend=levels((gapminder$continent)), pch=c(1:length(l
evels(gapminder$continent))), col=c(1:length(levels(y$continent))))
```

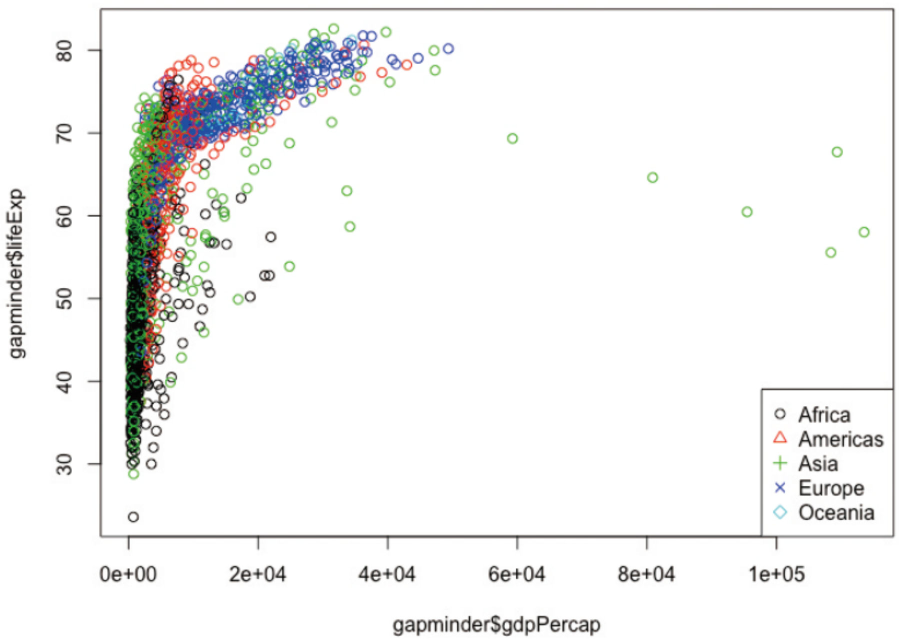


Figure 6-7 Graph for the intuitive understanding of data

Intuitive understanding of gapminder data (3)

- If it is not easy to observe because there are many samples in the lower range than the entire range of gdpPercap values, we can use the log scale to observe the samples evenly.

```
> plot(log10(gapminder$gdpPercap), gapminder$lifeExp, col=gapminder$continent)
> legend("bottomright", legend=levels((gapminder$continent)), pch=c(1:length(
levels(gapminder$continent))), col=c(1:length(levels(y$continent))))
```

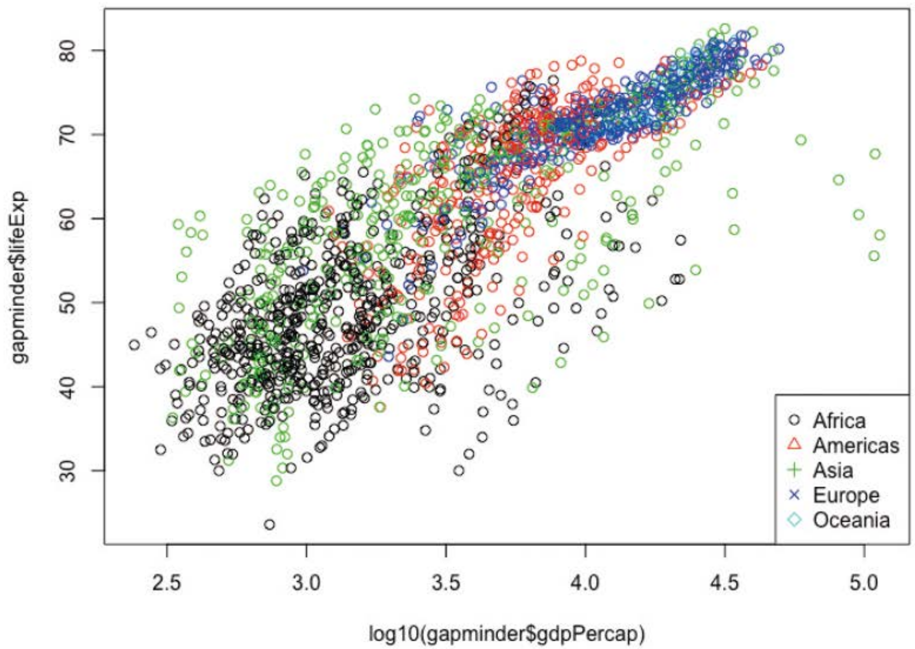


Figure 6-8 Graph using Log Scale

Intuitive understanding of gapminder data (4)

- Basic visualizations are possible using plot function of Base R,
- but the library dedicated to visualizations, ggplot2, makes it easier to specify additional options for graphs and achieve high-quality visualizations.

```
> library(ggplot2)
> ggplot(gapminder, aes(x=gdpPercap, y=lifeExp, col=continent))+geom_
point()+scale_x_log10()
```

Figure 6-9 Visualizing the data in [Figure 6-8] using ggplot function.

Intuitive understanding of gapminder data (5)

- ggplot function allows specifying the size of the plot marker relative to the population of each country by adding size =pop.
- Using the size option provided by gplot2, pop items can also be displayed on a single graph, making it easy to understand the interrelationships of various attributes.

```
> ggplot(gapminder, aes(x=gdpPercap, y=lifeExp, col=continent, size=pop)) +
geom_point() + scale_x_log10()
```

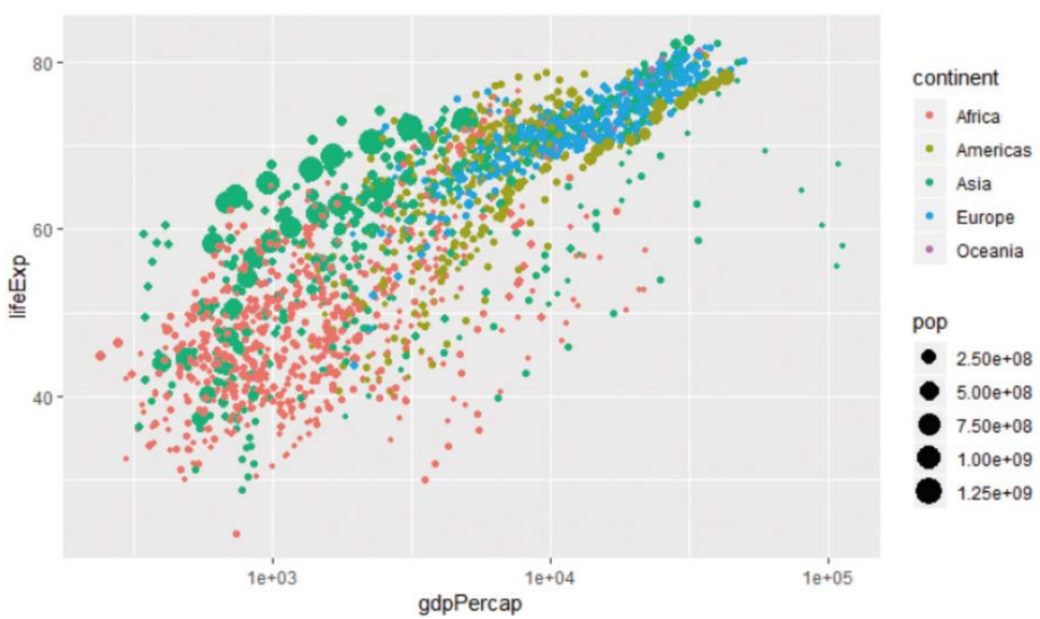


Figure 6-10 Marking pop variables in marker size.

Intuitive understanding of gapminder data (6)

```
> ggplot(gapminder, aes(x=gdpPercap, y=lifeExp, col=continent, size=pop)) +
geom_point(alpha = 0.5) + scale_x_log10()
```

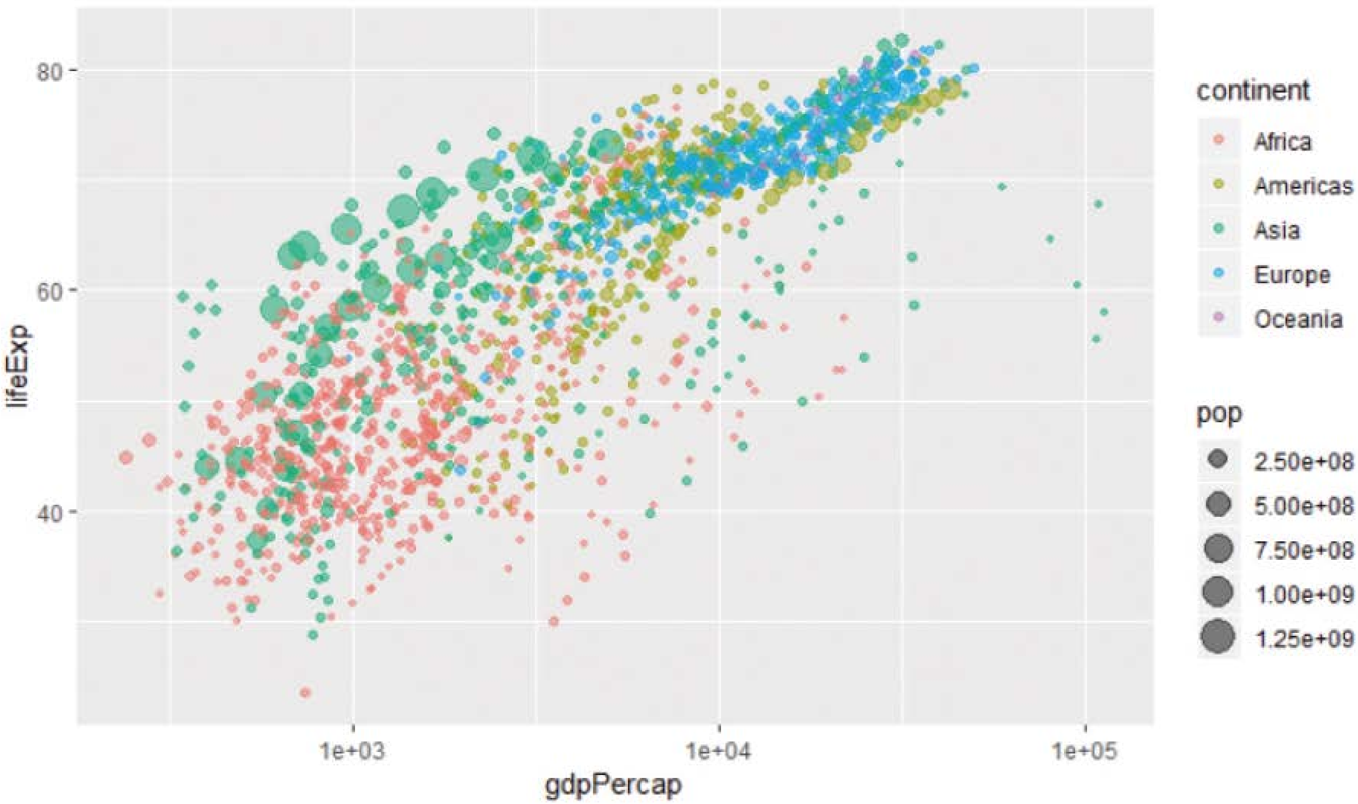


Figure 6-11 Displaying information using the transparency of markers

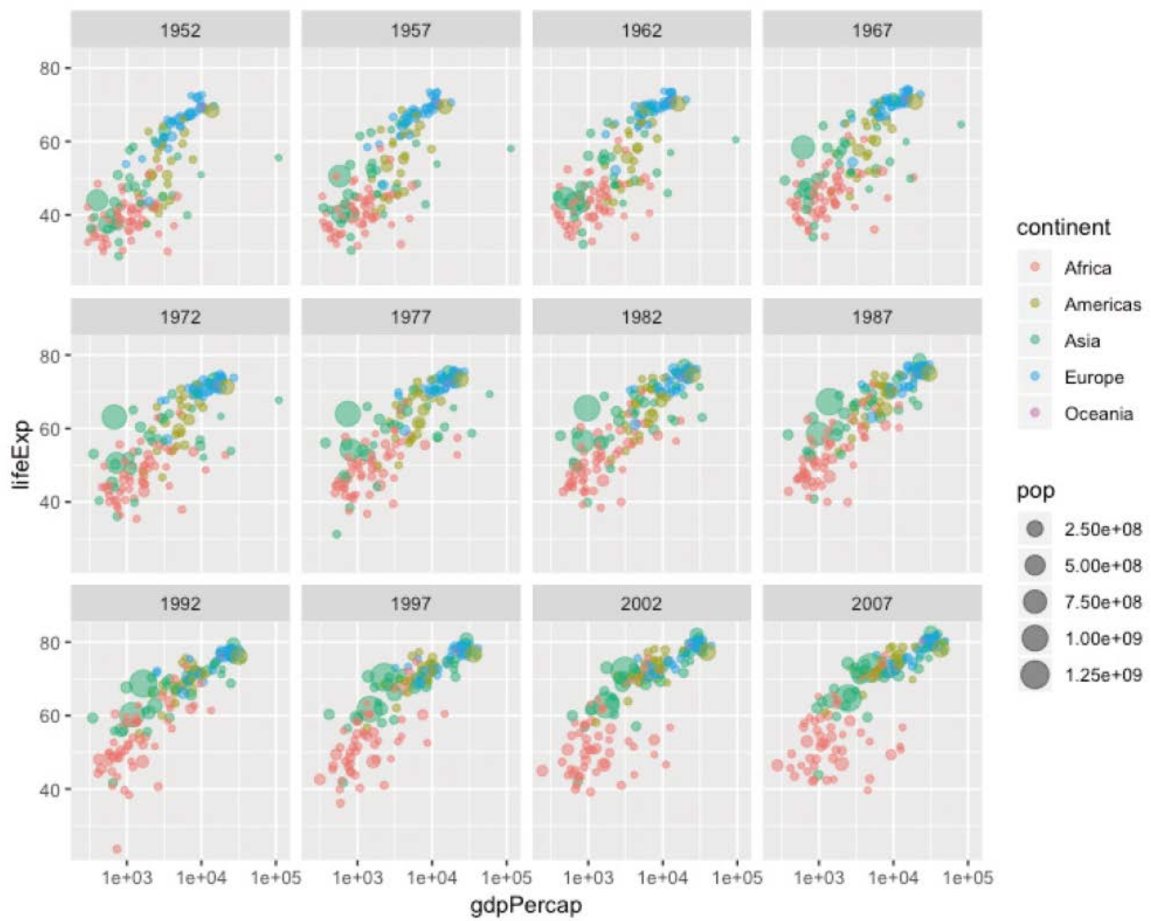
Intuitive understanding of gapminder data (7)

- It is recommended that the divided observation year be displayed separately in order to exquisitely visualize the data.
- filter function of the dplyr library can be used to extract data for each year in turn and draw graphs repeatedly,
- but the face_wrap function provided by ggplot2 can more simply replace programming for data processing and repetition.

02 Basic function of visualization: effective observation of large amounts of data

Intuitive understanding of gapminder data (8)

```
> ggplot(gapminder, aes(x=gdpPercap, y=lifeExp, col=continent, size=pop))+
geom_point(alpha=0.5)+scale_x_log10()+facet_wrap(~year)
```



It intuitively shows economic and welfare levels and changes in recent decades in several countries (continent) around the world included in the Gapminder data.

Figure 6-12 Graphs auto-generated separately by face_wrap function

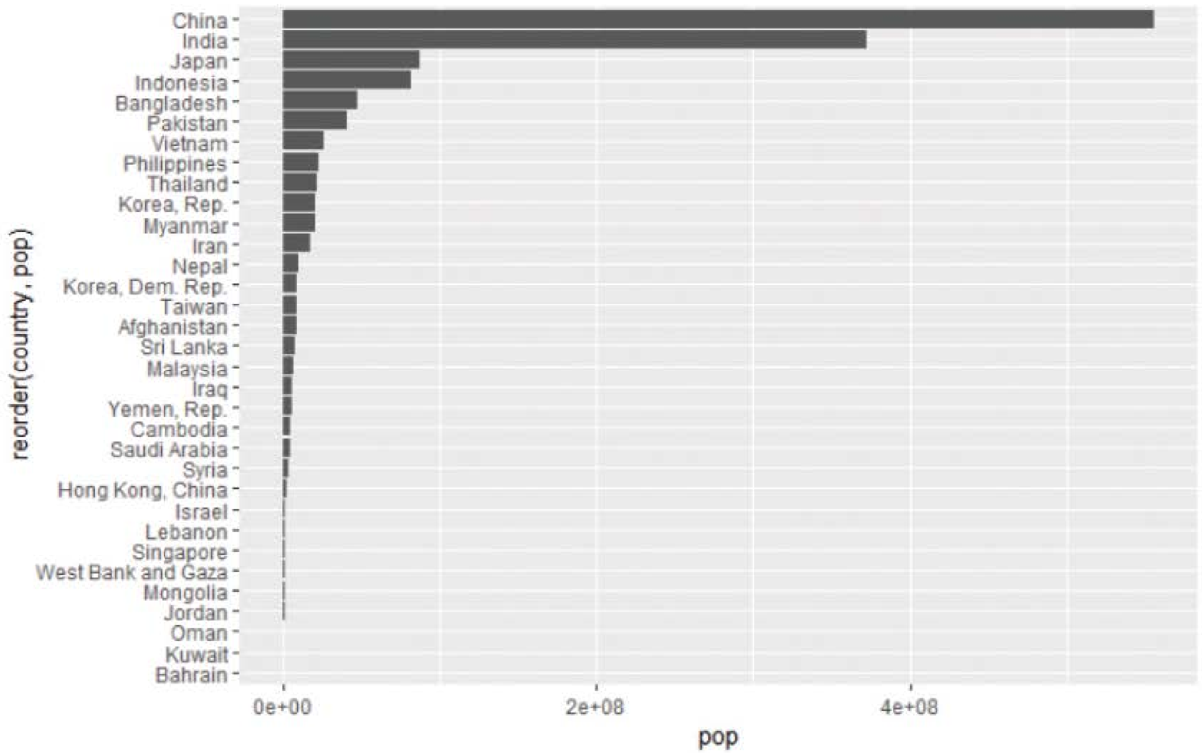
02 Basic features of visualizations : Observing data from multiple perspectives

- The common purpose of visualization is to clearly reveal the meaning inherent in the data, namely change· composition · distribution · correlation, etc.
- Because of the limitations of human cognitive abilities, it is impossible to identify at once the 'all changes in all variables' contained in the data.
- Therefore, the visualization of data should be tried repeatedly and from multiple perspectives. Changing the visualization view of the data gives insight into the various meanings contained in the data.
- Various visualizations are key technologies in data science.

Comparison/Ranking(1)

- Let's rank each country in the population distribution of the Asian continent in 1952.

```
> gapminder%>%filter(year==1952&continent=="Asia")%>%ggplot(aes(reorder(coun
try, pop), pop))+geom_bar(stat="identity")+coord_flip()
```



To solve the problem of overlapping country names when displayed on the horizontal axis, the position of the horizontal and vertical axes was changed using code_flip function.

Figure 6-13 A graph changed the position of the horizontal-vertical axis to accurately display country names

Comparison/Ranking(2)

- With the axis of the log scale, large values are converted to small and small values are converted to relatively large, allowing overall comparison on a single graph.

```
> gapminder %>% filter(year==1952 & continent== "Asia")%>% ggplot(aes(reorder(country,
pop), pop))+ geom_bar(stat="identity") + scale_y_log10() + coord_flip()
```

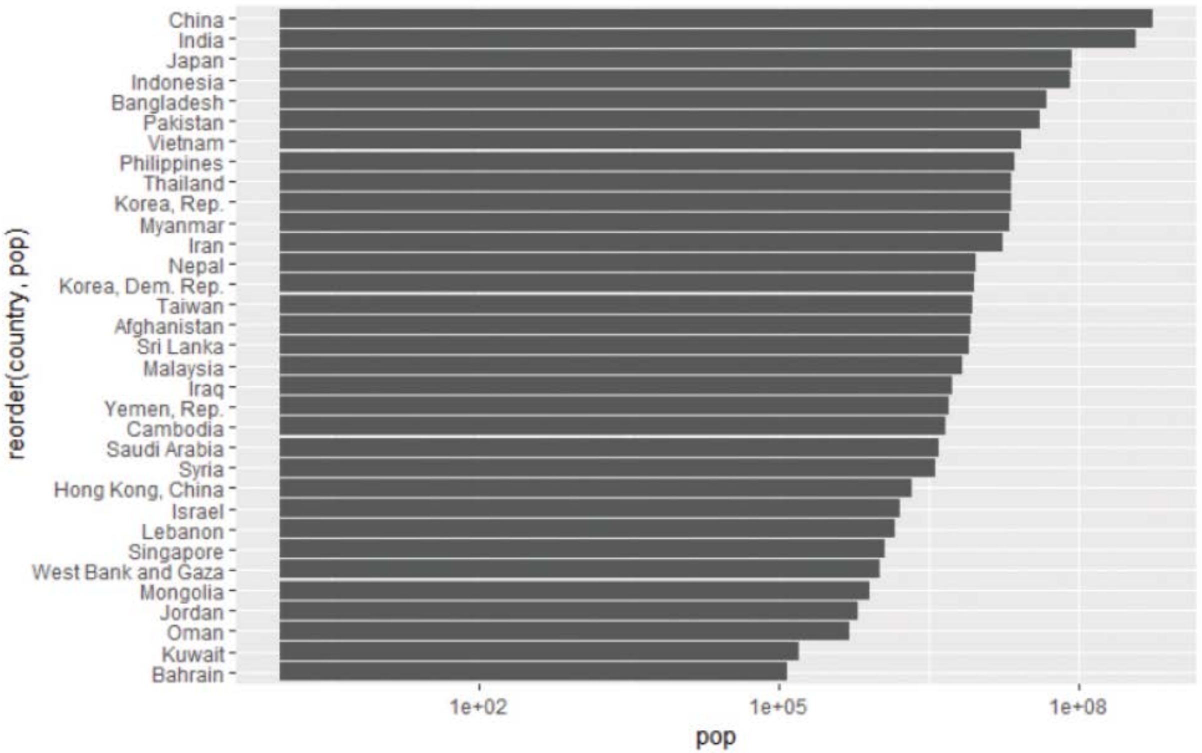


Figure 6-14 A logarithmic scale bar graph showing Asia's population ranking

Trend of changing(1)

- Let's visualize lifeExp changes of Korea in gapminder data by year.
- Use plots using dots and lines to show both the data values at the time of observation and the changes during the observation period.

```
> gapminder %>% filter(country == "Korea, Rep.") %>% ggplot(aes(year, lifeExp,
col = country)) + geom_point() + geom_line()
```

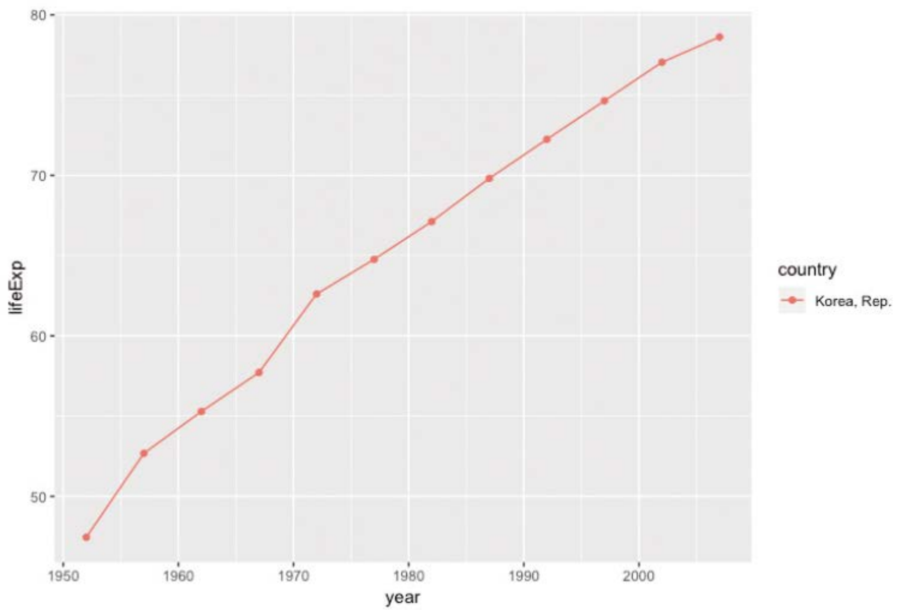


Figure 6-15 A graph visualizing population Changes in Korea with year

Trend of changing(2)

- When comparing changes in data at the same time, use color-coded multiple plots as However, it should be available for distinction using categorical attributes such as continent.
- We can also display the average trend line using geom_smooth function of ggplot2.

```
> gapminder %>% ggplot(aes(x=year, y=lifeExp, col=continent)) + geom_
point(alpha=0.2) + geom_smooth()
```

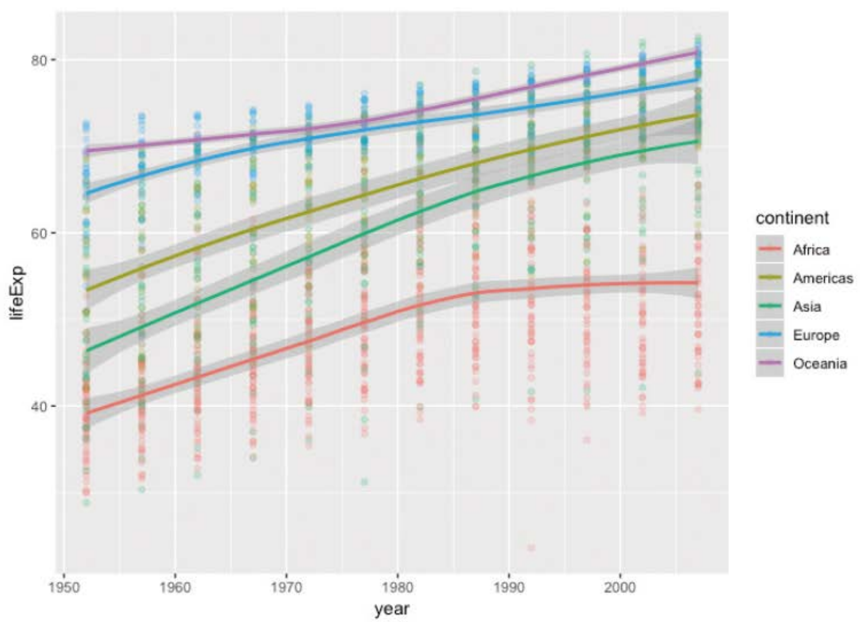


Figure 6-16 A Graph that added a trend line to the lifeExp change in each continent

Distribution or composition ratio(1)

- Let's visualize the distribution of lifeExp worldwide in 1952.
  - Use hist function provided by Base R.

```
> x = filter(gapminder, year == 1952)
> hist(x$lifeExp, main = "Histogram of lifeExp in 1952")
```

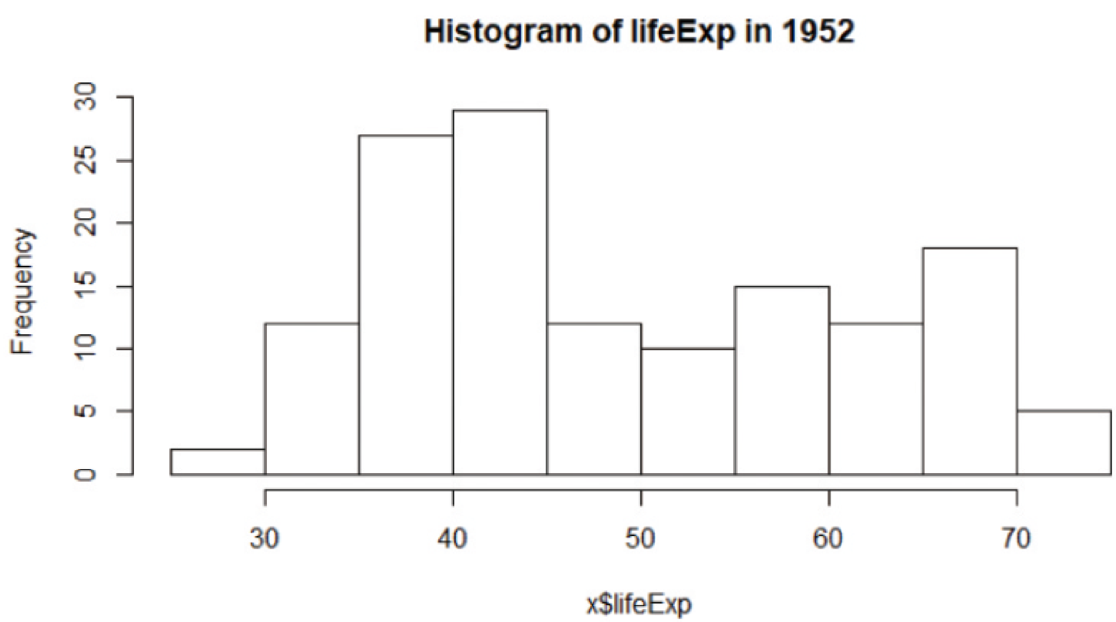


Figure 6-17 Distribution histogram of lifeExp in 1952(Use default host function in R)

Distribution or composition ratio(2)

- Let's visualize the distribution of lifeExp worldwide in 1952.
  - ggplot function can be used to indicate as follows.

```
> x %>% ggplot(aes(lifeExp)) + geom_histogram()
```

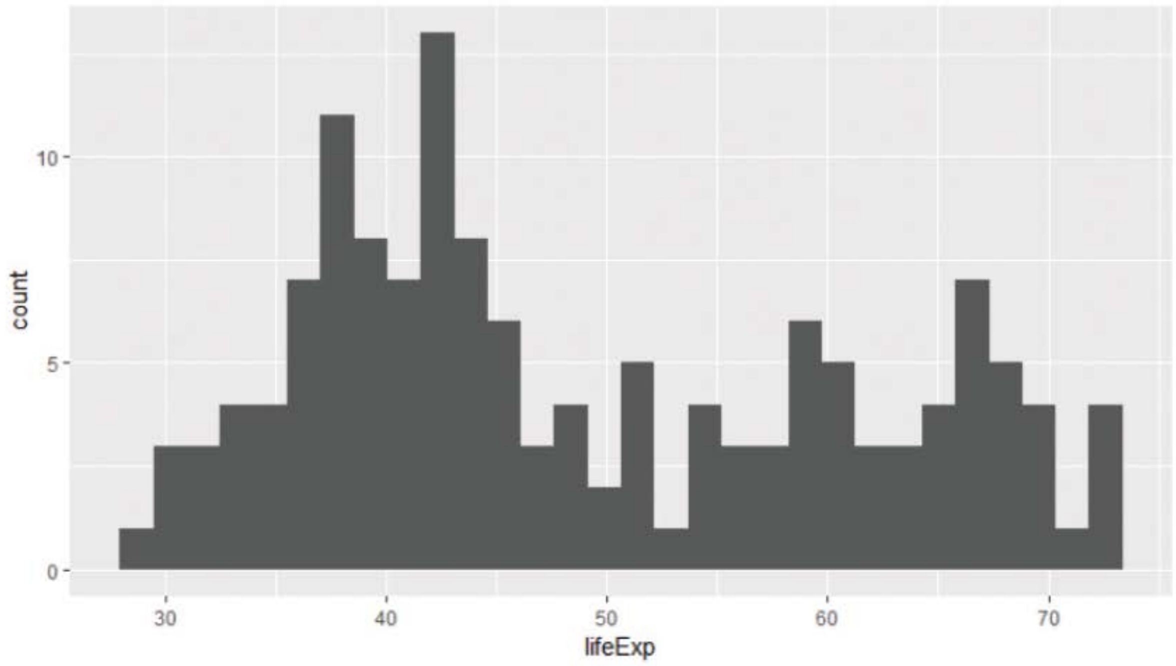
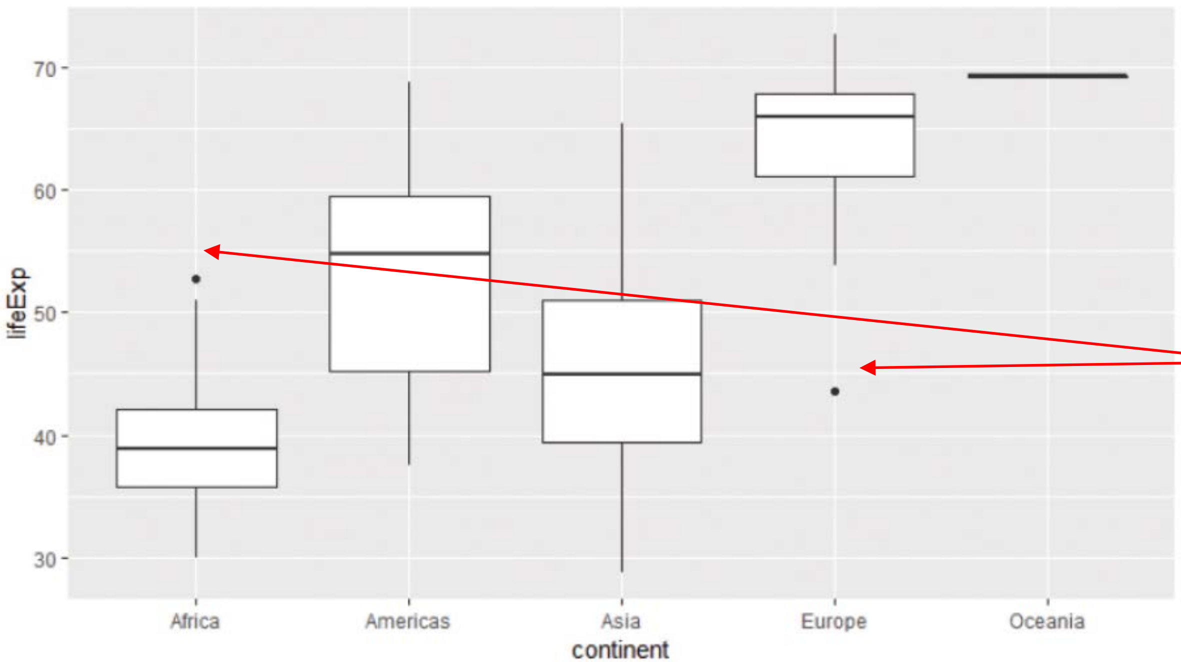


Figure 6-18 Distribution histogram of lifeExp in 1952(Use ggplot function)

Distribution or composition ratio(3)

- Both of the preceding graphs show a distribution that aggregates the life expectancy of all countries in the year, regardless of country or continent.
- boxplot function allows you to look at the distribution characteristics of each continent at the same time.

```
> x %>% ggplot(aes(continent, lifeExp)) + geom_boxplot()
```



Points are data that deviate from the normal distribution, which can be removed through the data refining process, boxplot is also used to identify and remove abnormal values.

Figure 6-19 Result of visualizing the distribution of continents at the same time using boxplot

Correlation(1)

- One of the key tasks in the data analysis process is to find the correlation between attributes
- Because correlation is not only used to describe the meaning and causality inherent in the data but also used to predict unknown outcomes through modeling.

```
> plot(log10(gapminder$gdpPercap), gapminder$lifeExp)
```

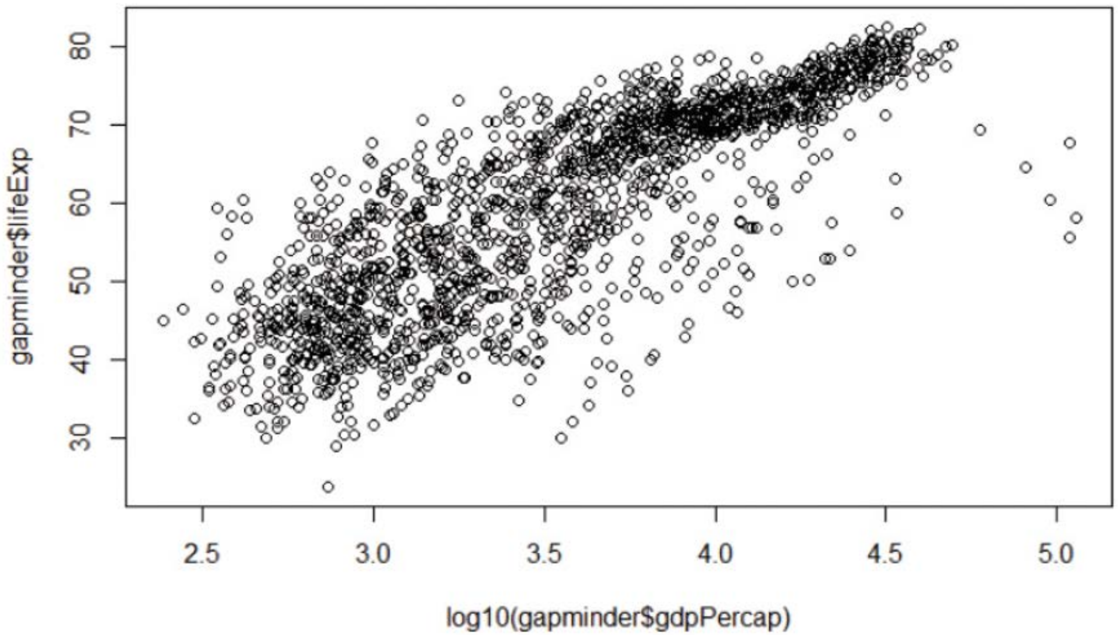


Figure 6-20 Visualization to check the correlation between lifeExp and gdpPercap

03 Visualization tool

- The basic visualization functions into R are relatively easy to obtain effective visualization results.

- However, there are many things you should know, to create graphs using additional options.

- So it is convenient to use in a simple visualization process, not a big problem.

03 Visualization tool : Base R

plot function(1)

- It is the most common graph visualization function that enables several types of plots, such as straight lines, points, etc.
- Use cars data embedded in Base R to check the basic command format and visualization techniques.

```
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10

# type="p" is the point plot, main="cars" is the title of the graph
> plot(cars, type = "p", main = "cars")
```

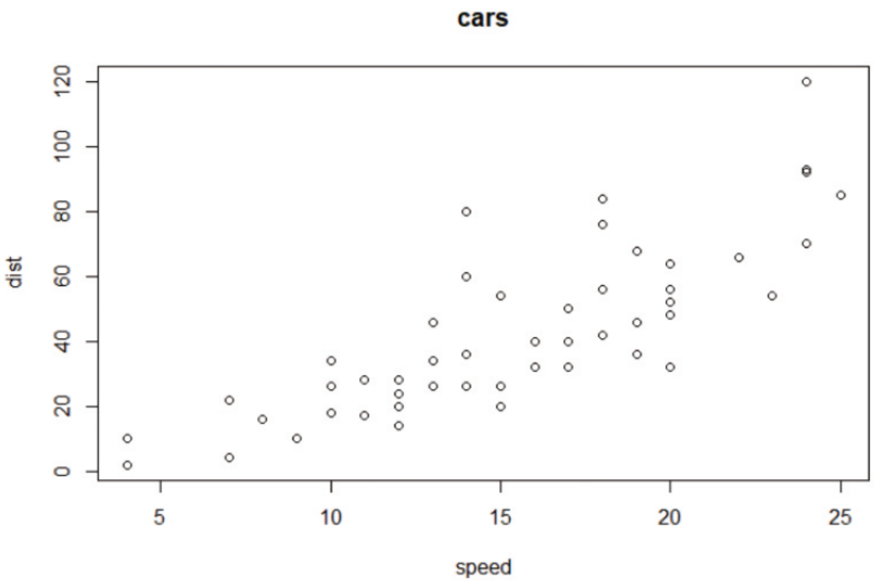


Figure 6-21 Basic point graph using plot function

03 Visualization tool : Base R

plot function(2)

```
> plot(cars, type="l", main="cars")          # type="l" is the plot using line
```

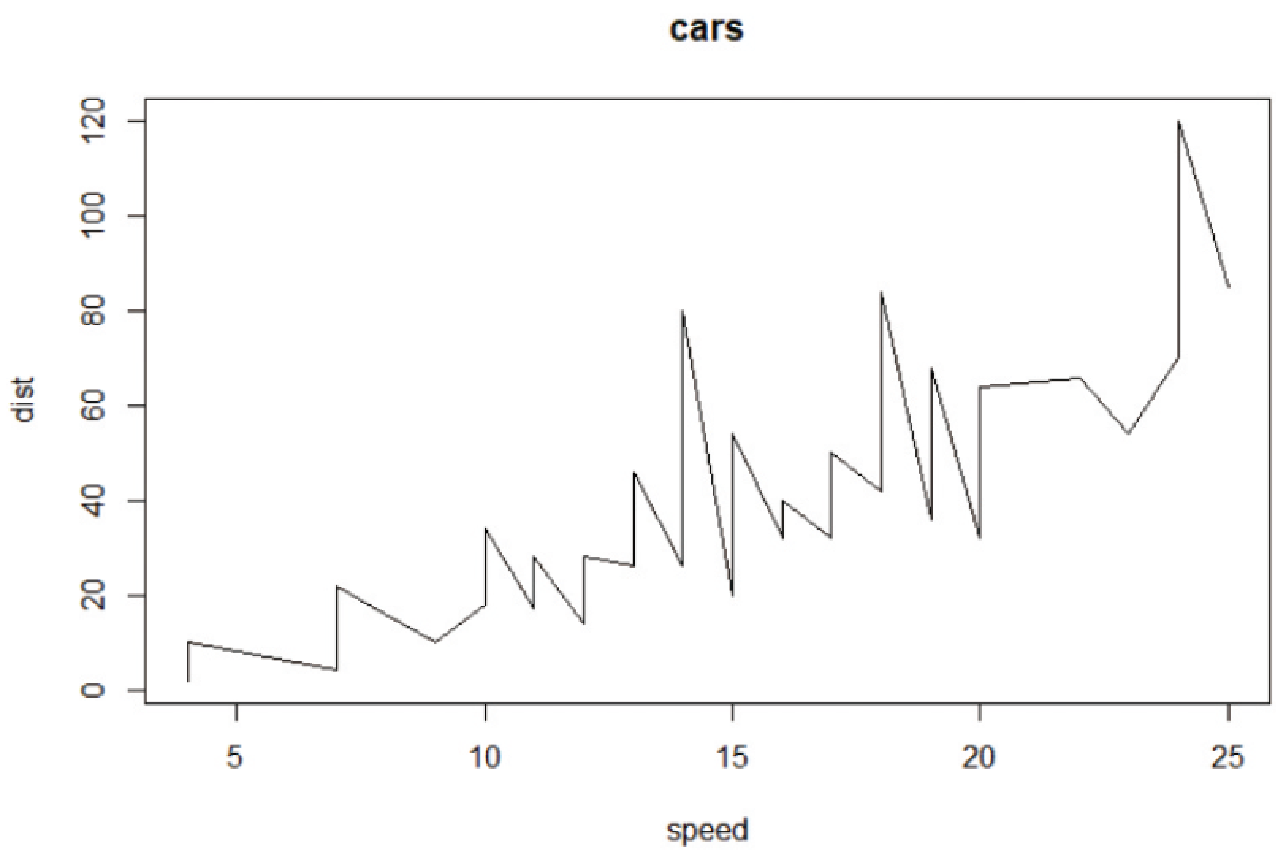


Figure 6-22 Basic line graph using plot function

03 Visualization tool : Base R

plot function(3)

```
> plot(cars, type="b", main="cars")  # type="b" is the plot using with both point and line
```

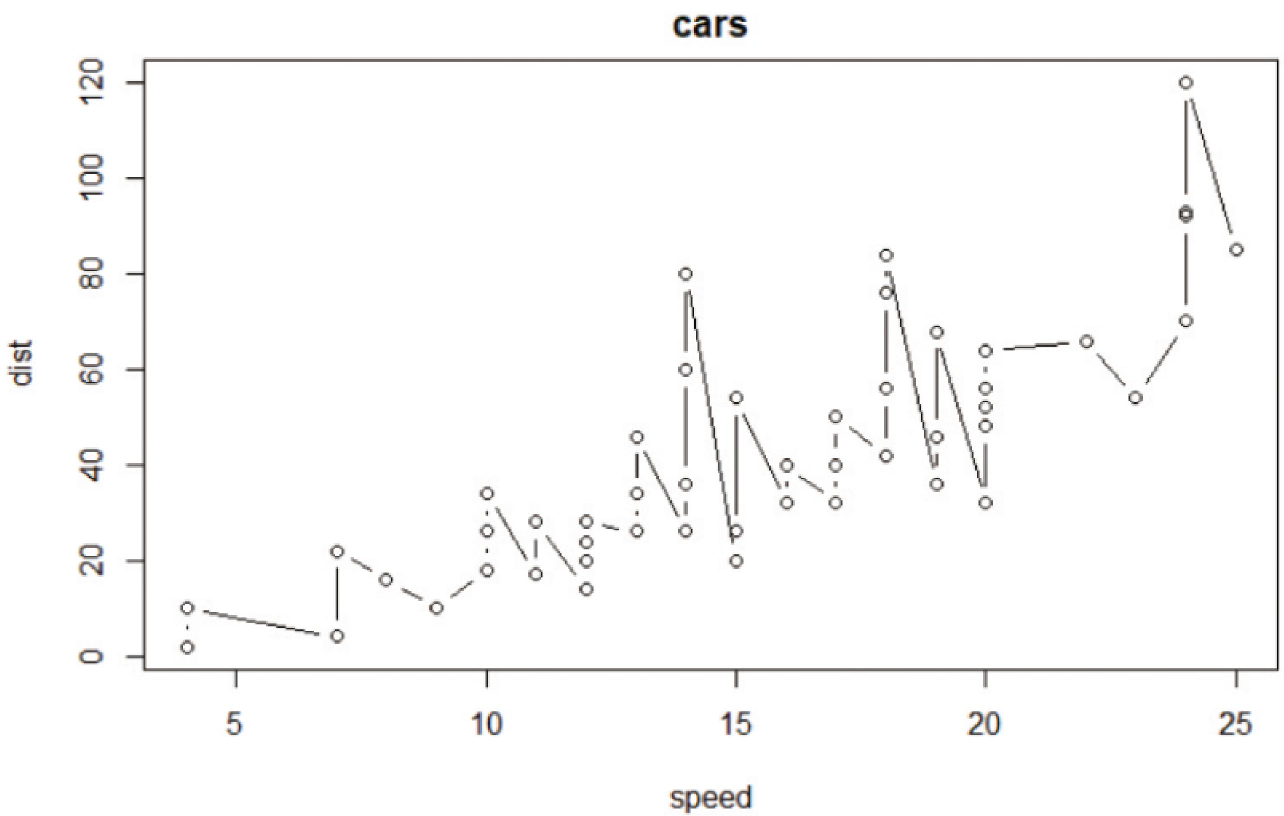


Figure 6-23 Point and line graph using plot function

03 Visualization tool : Base R

plot function(4)

```
> plot(cars, type="h", main="cars") # type="h" is the bar graph, such as a histogram
```

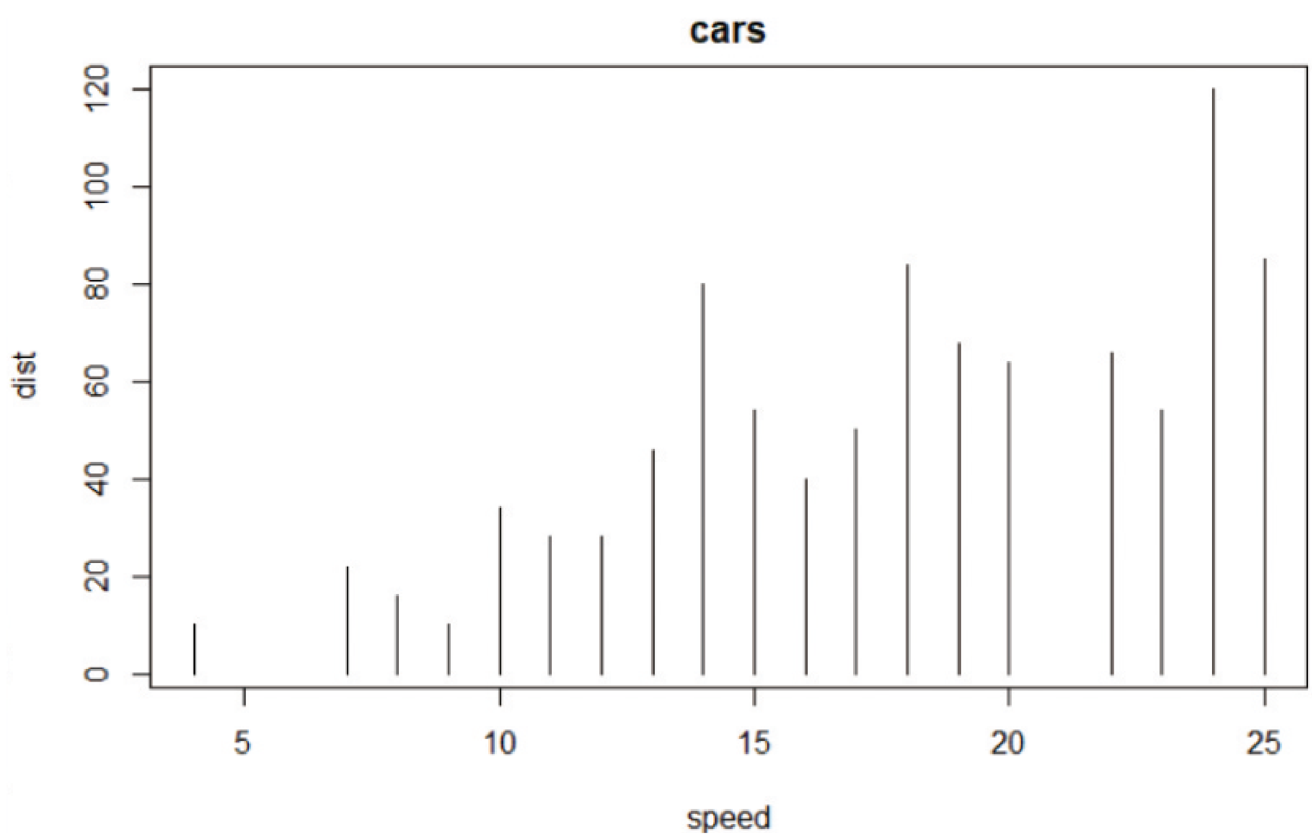
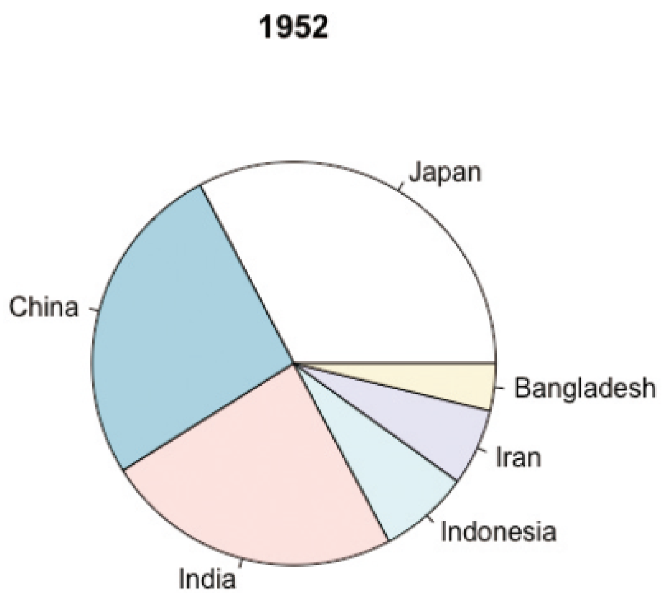


Figure 6-24 Bar graph using plot function

03 Visualization tool : Base R

pie · barplot function(1)

```
> x = gapminder %>% filter(year == 1952 & continent == "Asia") %>% mutate(gdp =
gdpPercap*pop) %>% select(country, gdp) %>% arrange(desc(gdp)) %>% head()
> pie(x$gdp, x$country)
> barplot(x$gdp, names.arg = x$country)
```



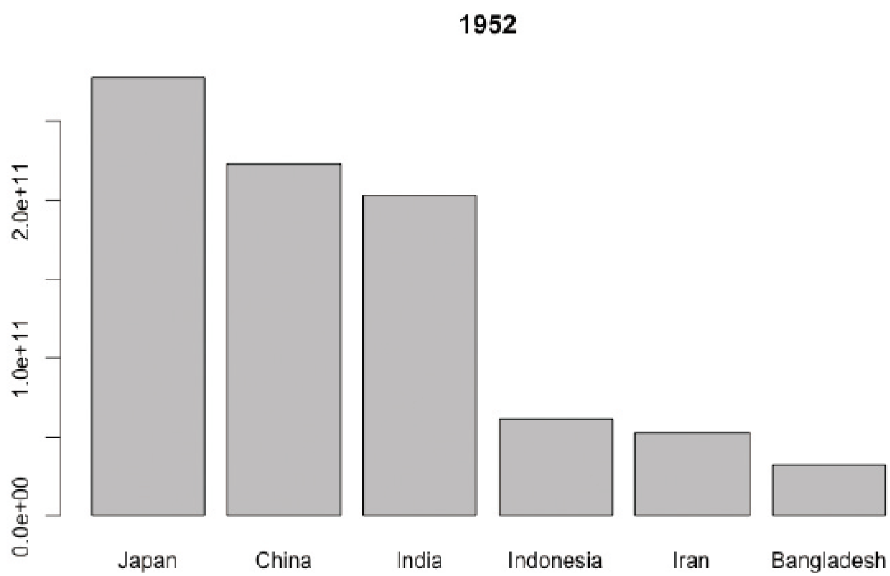


Figure 6-25 Composition and ranking of gdp in Asian countries in 1952 visualized using pie and barplot functions

03 Visualization tool : Base R

pie · barplot function(2)

```
> x = gapminder %>% filter(year == 2007 & continent == "Asia") %>% mutate(gdp =
gdpPercap*pop) %>% select(country, gdp) %>% arrange(desc(gdp)) %>% head()
> pie(x$gdp, x$country)
> barplot(x$gdp, names.arg = x$country)
```

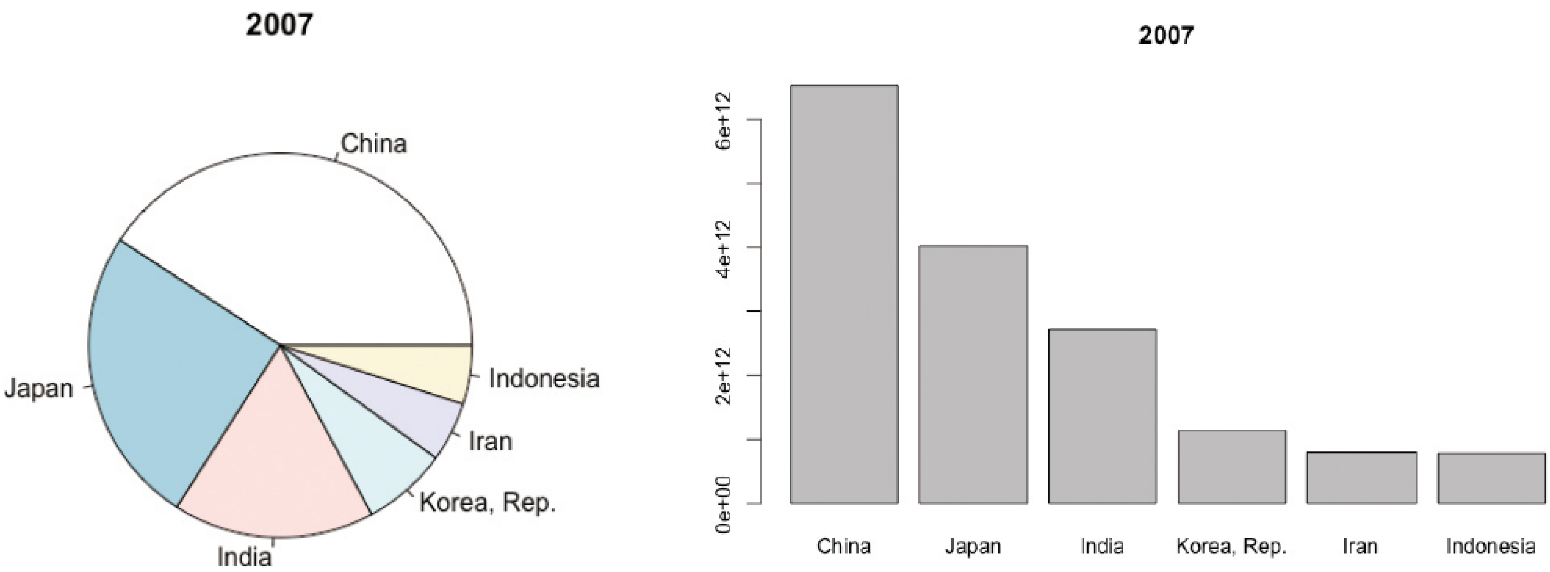


Figure 6-26 Composition and ranking of gdp in Asian countries in 2007 visualized using pie and barplot functions

03 Visualization tool : Base R

- matplot function

```
> matplot(iris[, 1:4], type = "l")
> legend("topleft", names(iris)[1:4], lty = c(1, 2, 3, 4), col = c(1, 2, 3, 4))
```

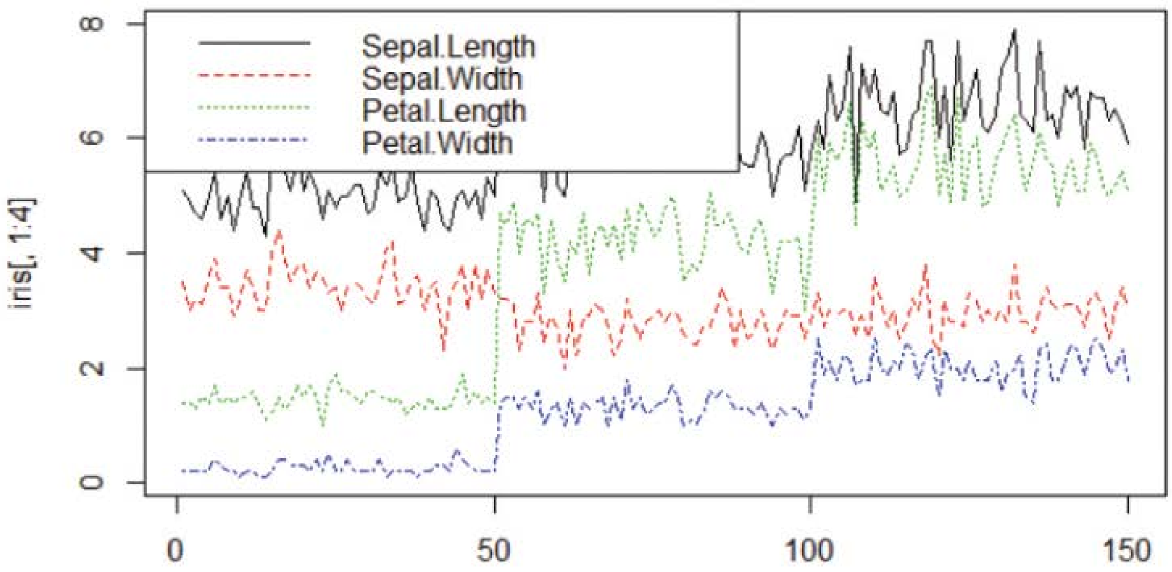


Figure 6-27 Multiple-plot using the matplot function

03 Visualization tool : Base R

- hist function

```
> hist(cars$speed)
```

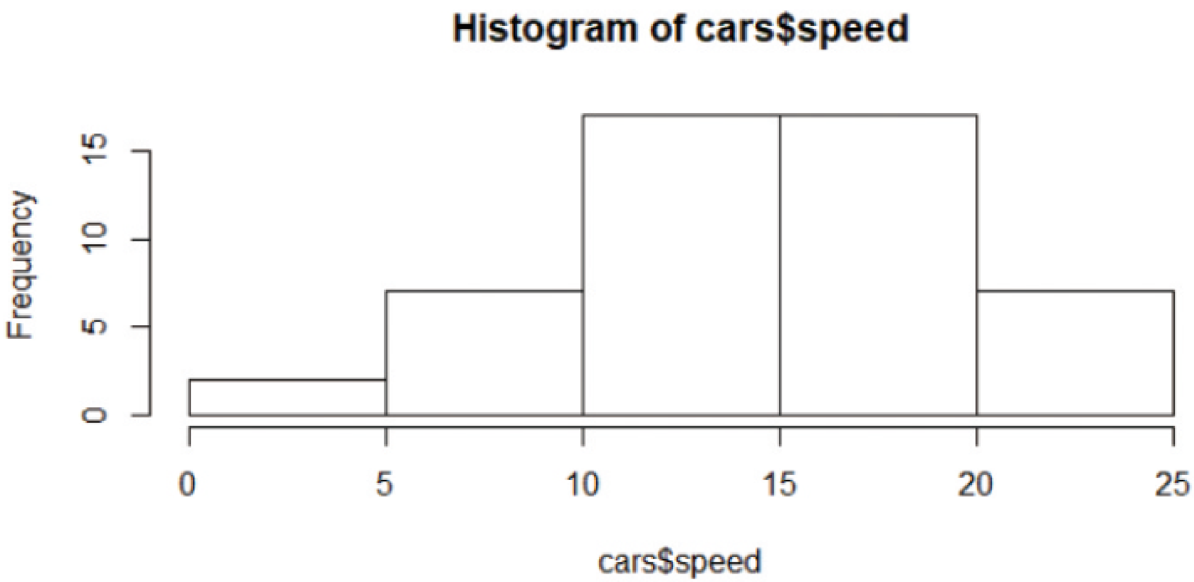


Figure 6-28 histogram using host function

03 Visualization Tool : ggplot2 library specialized in visualization

- ggplot2 is the most popular visualization library
  - gg is the abbreviation for grammar of graphics.
  - Because ggplot2 contains systematic visualization commands, it has made the data visualization task more intuitive and efficient.
- Functions of ggplot2 library follow the following basic expressions, consisting of three elements:

```
> ggplot(gapminder, aes(x=gdpPercap, y=lifeExp, col=continent)) + geom_
point(alpha=0.2)
```

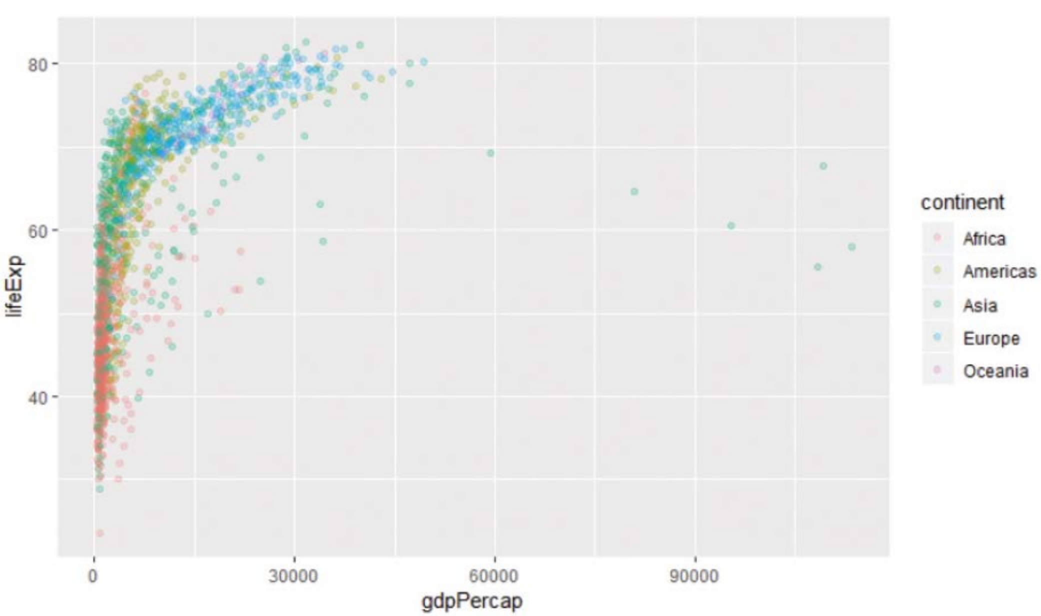


Figure 6-29 Visualization using geom_point function of ggplot2 library

03 Visualization Tool : ggplot2 library specialized in visualization

ggplot2 function

- It works to create a visualization object.
  - The input data, and items corresponding to the horizontal and vertical axes should be specified in the initialization process.
  - Designating using aes inside.

03 Visualization Tool : ggplot2 library specialized in visualization

geom_point function(1)

- Draw a plot to display the data as points.
  - Inside, the alpha option allows setting the opacity of the points (transparent 0.0 to opaque 1.0), so that we can see the distribution and frequency of the data even if the markers are overlaid.
- Additionally available functions
  - geom_line : Displays data using line.
  - geom_bar : Displays the data using bar graph. Without a separate setting, it will automatically calculate the distribution and draw a histogram in the same way as the geom_histogram, so if you want to draw a non-histogram bar graph, i.e., a graph with both x and y specified in the aes function, you should specify stat="identity" option.
  - geom_histogram : It's plot function dedicated to a histogram.
  - The default option is to stack the bars up with position ="stack". Specify the position ="dodge" option to display bars side by side.

03 Visualization Tool : ggplot2 library specialized in visualization

geom_point function(2)

```
> gapminder %>% filter(year == 2007) %>% ggplot(aes(lifeExp, col=continent)) +
geom_histogram()
```

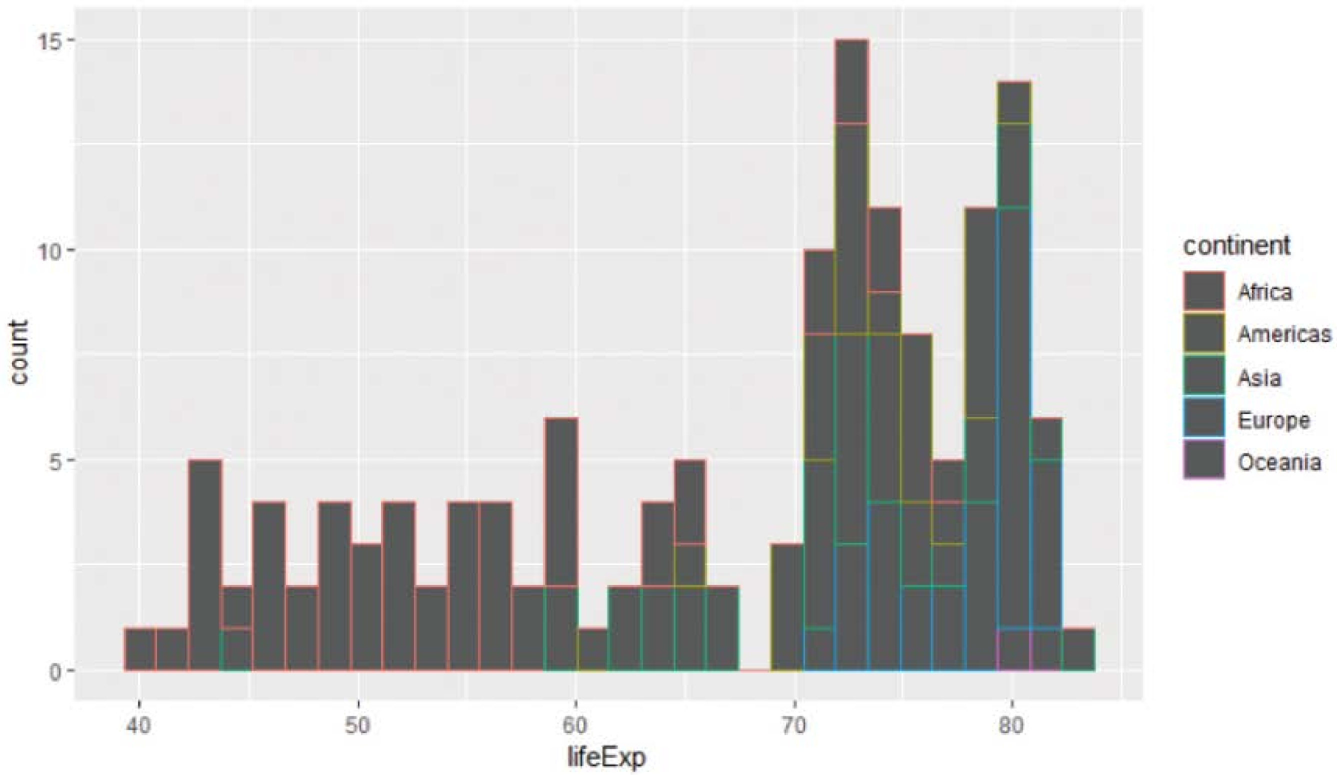


Figure 6-31 Histogram using geom_histogram function : showing the distribution of groups stacked vertically

03 Visualization Tool : ggplot2 library specialized in visualization

geom_point function(3)

```
> gapminder %>% filter(year == 2007) %>% ggplot(aes(lifeExp, col = continent)) +
geom_histogram(position = "dodge")
```

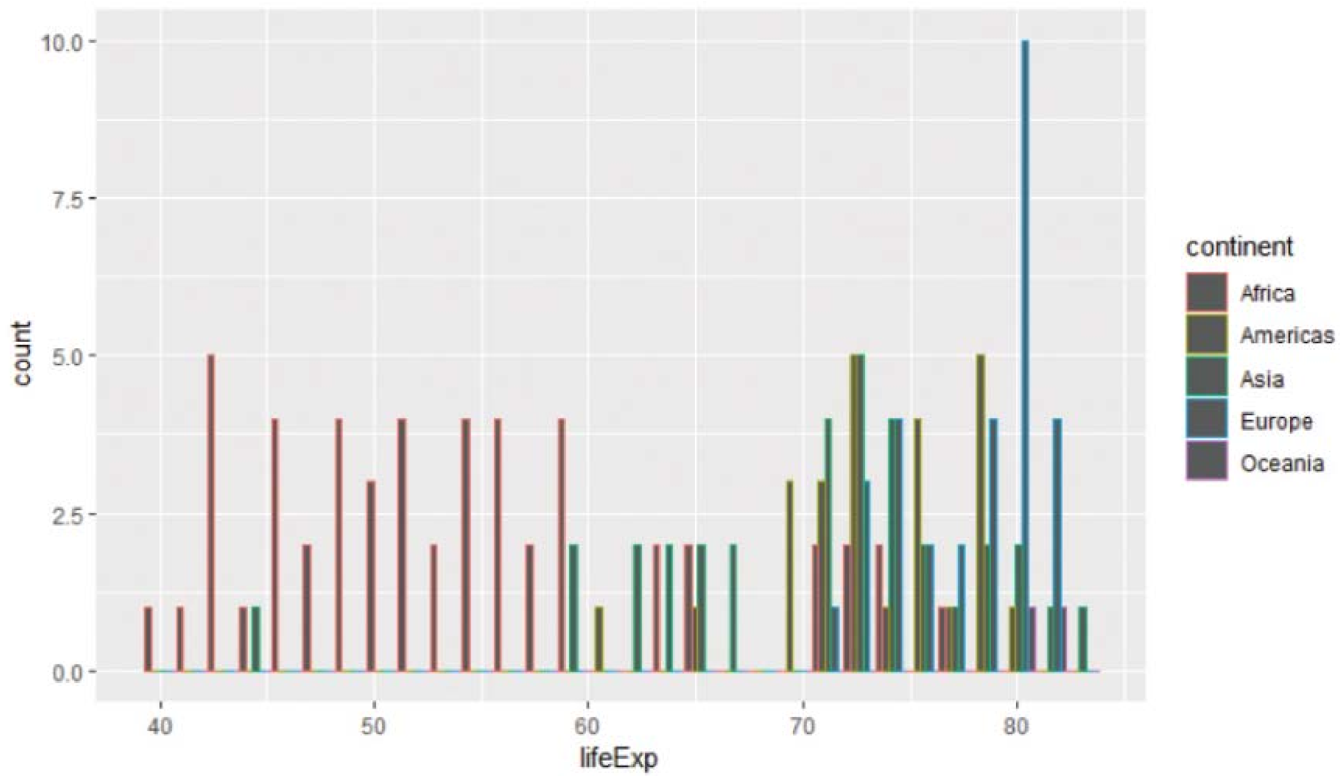


Figure 6-32 Histogram using geom_histogram function : showing the distribution of groups horizontally

03 Visualization Tool : ggplot2 library specialized in visualization

geom_boxplot function

- It is a function that observes the distribution of multiple items at once and is useful for identifying outliers.

```
> gapminder %>% filter(year == 2007) %>% ggplot(aes(continent,lifeExp, col=
continent)) + geom_boxplot()
```

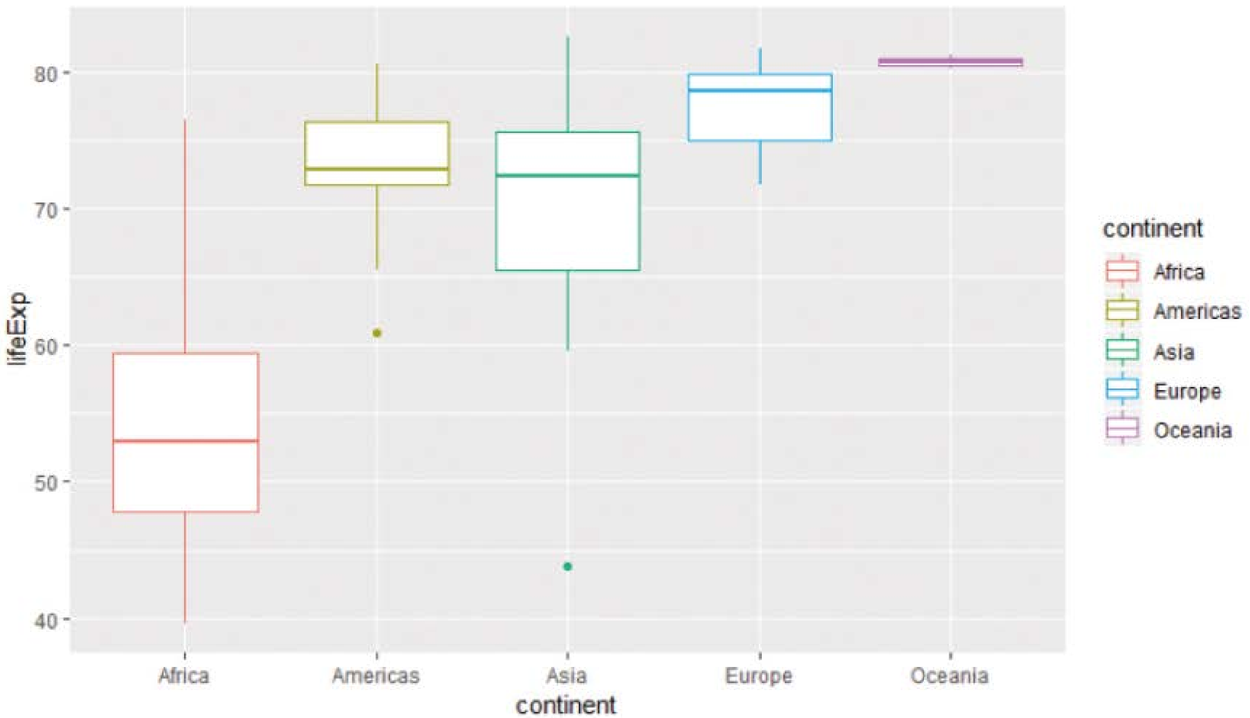


Figure 6-33 boxplot using geom_boxplot function

03 Visualization Tool : ggplot2 library specialized in visualization

scale_x_log10 · scale_y_log10 function

- Using scale_x_log10 and scale_y_log10 functions, we can achieve the same effect by changing the scale of the axis without having applied log directly to the data.

```
> ggplot(gapminder, aes(x=gdpPercap, y=lifeExp, col=continent)) + geom_
point(alpha=0.2)

> ggplot(gapminder, aes(x=gdpPercap, y=lifeExp, col=continent)) + geom_
point(alpha=0.2) + scale_x_log10()     # Converting horizontal axis using log scale
```

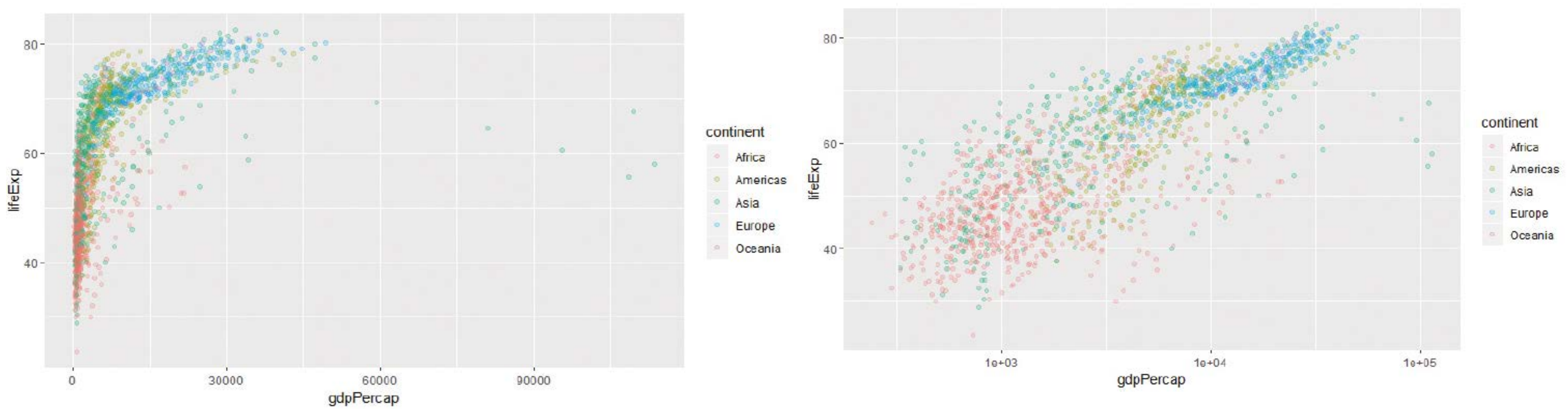
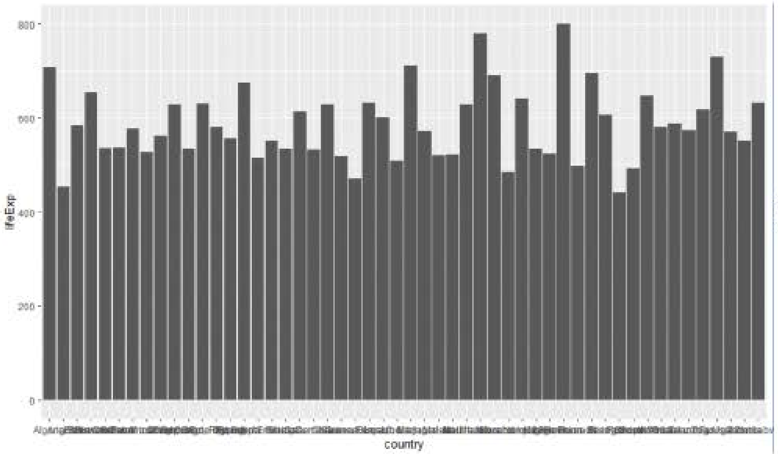


Figure 6-34 Graph applied log10 scale to the horizontal axis using scale_x_log10 function

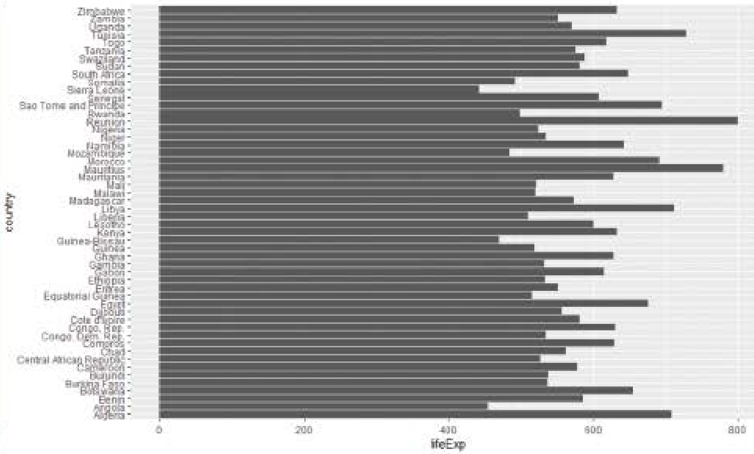
03 Visualization Tool : ggplot2 library specialized in visualization

coord_flip function

```
> gapminder %>% filter(continent == "Africa") %>% ggplot(aes(country, lifeExp)) +
geom_bar(stat = "identity")                        # [Figure 6-35(a)]

> gapminder %>% filter(continent == "Africa") %>% ggplot(aes(country, lifeExp)) +
geom_bar(stat = "identity") + coord_flip()        # [Figure 6-35(b)] Changing the direction of a plot
```



(a) Basic bar graph



(b) Bar graph of changed the horizontal and vertical axes

Figure 6-35 Bar graph

scale_fill_brewer function(1)

- When using the col option inside aes function,
  - The colors used depend on the color palette.
  - Using scale_fill_brewer function, we can change the color palette on the screen by choosing from among the different combinations of color palettes.
  - Using RColorBrewer library together, we can choose from a range of color palettes that are much more diverse than the basic color scheme of R.

03 Visualization Tool : ggplot2 library specialized in visualization

scale_fill_brewer function(2)

```
> library(RColorBrewer)
> display.brewer.all()
```

Figure 6-36 Name and color scheme of various color palettes provided by RColorBrewer library

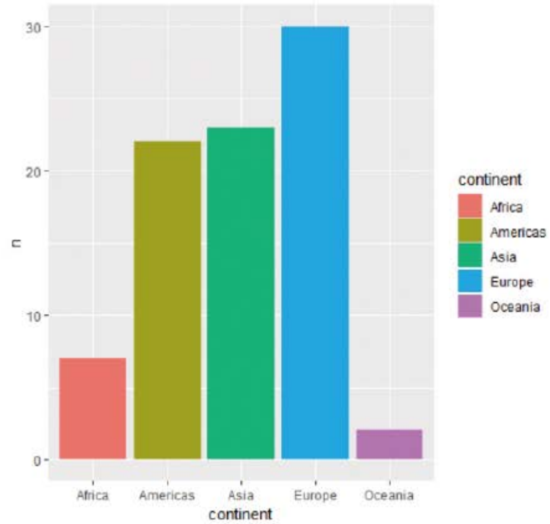
03 Visualization Tool : ggplot2 library specialized in visualization

scale_fill_brewer function(3)

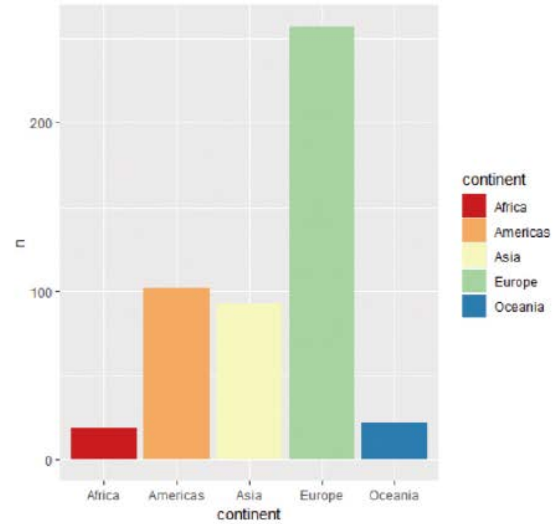
```
> # [Figure 6-37(a)] :  Graph using Default Palette
> gapminder %>% filter(lifeExp>70) %>% group_by(continent) %>% summarize(n
=n_distinct(country)) %>% ggplot(aes(x=continent, y=n)) + geom_bar(stat=
"identity", aes(fill=continent))

> # [Figure 6-37(b)] :  Graph using Spectral Palette
> gapminder %>% filter(lifeExp>70) %>% group_by(year, continent) %>% summarize(n
= n_distinct(country)) %>% ggplot(aes(x = continent, y = n)) + geom_bar(stat =
"identity", aes(fill = continent)) + scale_fill_brewer(palette = "Spectral")

#  [Figure 6-37(c)] : Graph using Blues Palette
> gapminder %>% filter(lifeExp>70) %>% group_by(continent) %>% summarize(n
=n_distinct(country)) %>% ggplot(aes(x=continent, y=n)) + geom_bar(stat=
"identity", aes(fill=continent)) + scale_fill_brewer(palette="Blues")

# [Figure 6-37(d)] :  Graph using Oranges Palette
> gapminder %>% filter(lifeExp>70) %>% group_by(continent) %>% summarize(n
=n_distinct(country)) %>% ggplot(aes(x=continent, y=n)) + geom_bar(stat=
"identity", aes(fill=continent)) + scale_fill_brewer(palette="Oranges")
```

03 Visualization Tool : ggplot2 library specialized in visualization

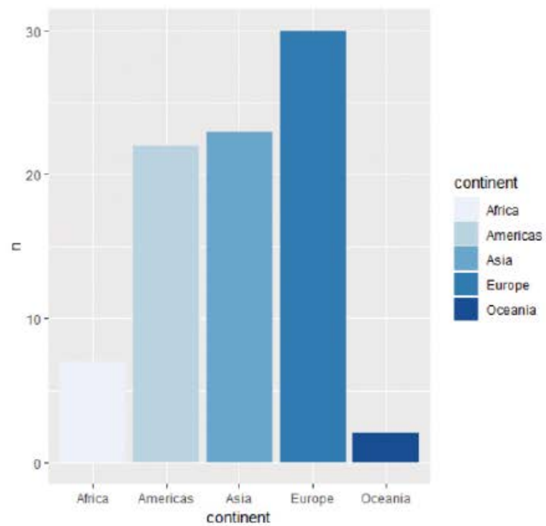
- scale_fill_brewer function(4)



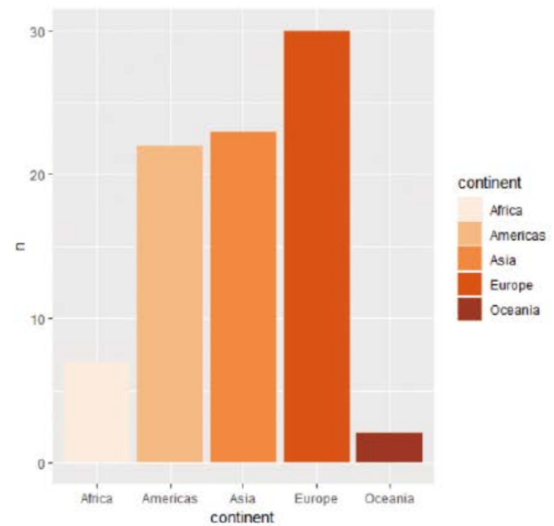
(a) Default Palette



(b) Spectral Palette



(c) Blues Palette



(d) Oranges Palette

Figure 6-37 Graphs using Color Palette

03 Visualization Tool : ggplot2 library specialized in visualization

scale_fill_brewer function(5)

- Use reorder function to adjust the order of the data displayed on the graph.

```
> # reorder (contents, -n) means that continent should be sorted in descending order by n
> gapminder %>% filter(lifeExp>70) %>% group_by(continent) %>% summarize(n=n_
distinct(country)) %>% ggplot(aes(x=reorder(continent, -n), y=n)) + geom_
bar(stat="identity", aes(fill=continent))+scale_fill_brewer(palette="Blues")
```

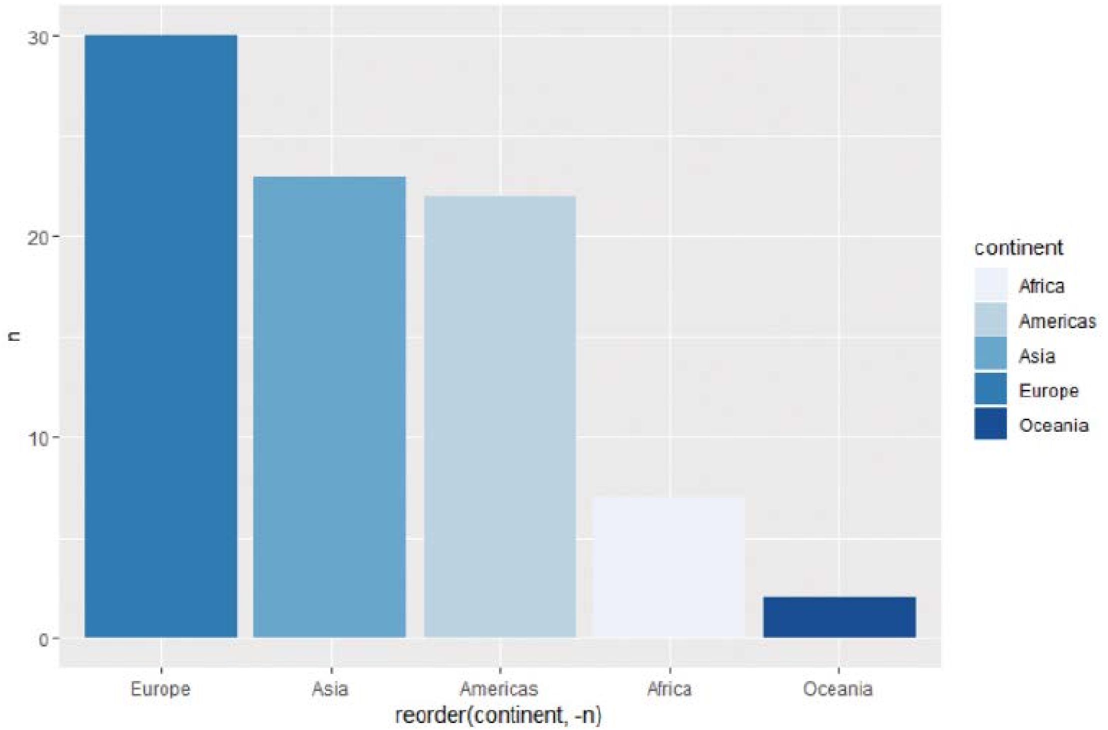
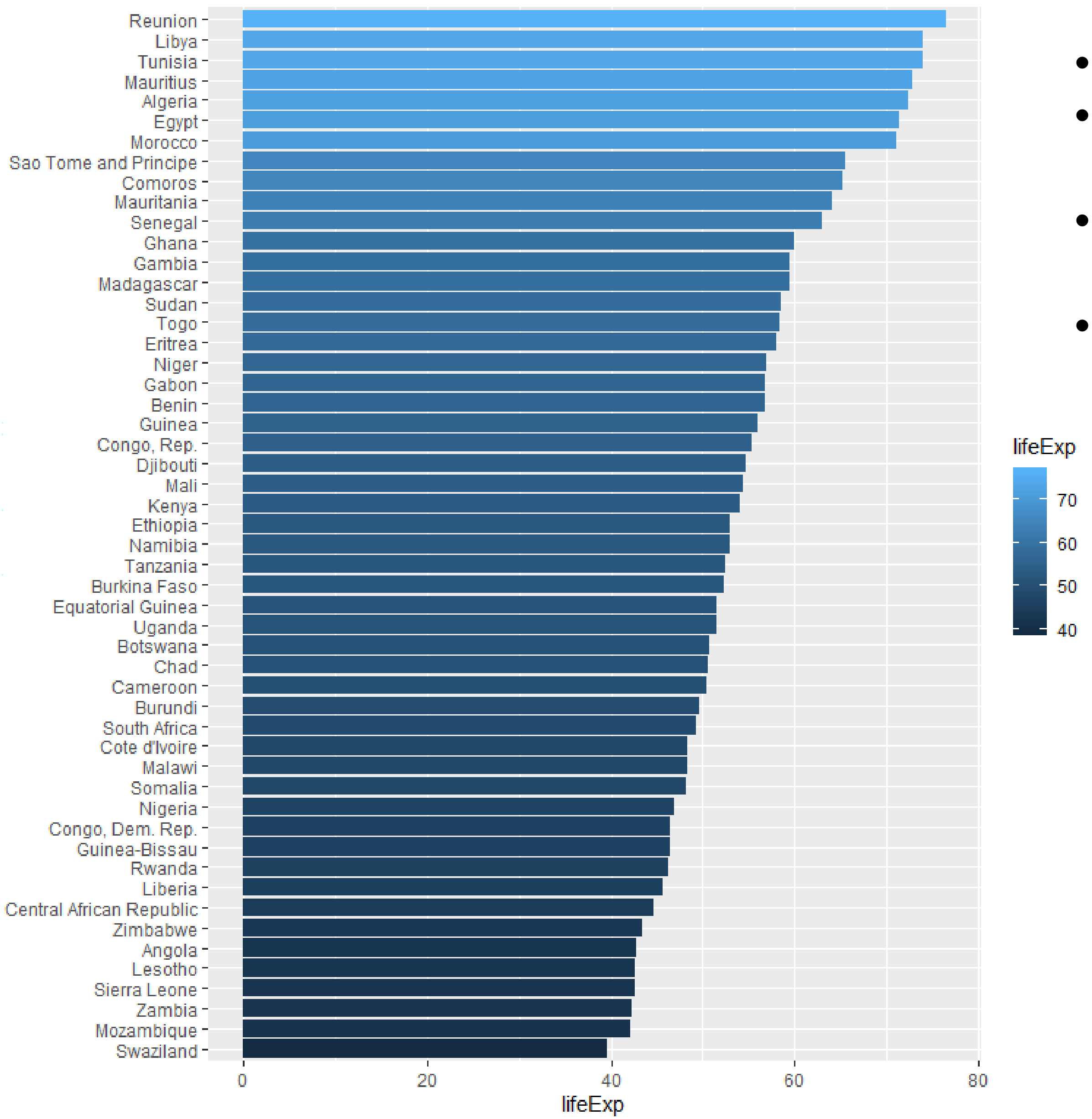


Figure 6-38 Graph using order sort and continuous color palettes to suit the ranking visualization

Practice in-class



- Filter: Africa, 2007
- Reorder: lifeExp, descending order
- Fill=gradient by lifeExp
- coord_flip

04 Exploring data using visualization

- Visualization often provides insight that cannot be obtained in any other way.
- Visual exploration is becoming a much more interesting task than in the past, especially thanks to effective data analysis tools such as R.
- In the visual exploration phase, we tend to visualize data as widely as possible because we are unsure of what the ideas we want to find are, and sometimes we use a combination of data.
- Thanks to efficient tools, sometimes we excessively put effort into creating a good chart.
- However, the essence of visualization is a visual exploration, which discovers and analyzes the meaning of the data.

04 Exploring data using visualization

Visual Exploration of gapminder Data (1)

- Data show the level of economy and welfare in many countries around the world, commonalities and differences that appear in different countries or regions, and the noticeable economic growth of some Asian countries over the past decades.
- Furthermore, by analyzing the interrelationships among attributes such as year, constant, country, pop, gdpPercap, lifeExp, etc., we can think about the causes of these differences and changes.

04 Exploring data using visualization

Visual Exploration of gapminder Data (2)

- Let's use R to visualize the economic and welfare levels and their changes in many countries around the world.

```
> gapminder %>% ggplot(aes(gdpPercap, lifeExp, col=continent)) + geom_
point(alpha=0.2) + facet_wrap(~year) + scale_x_log10()
```

The link between ggplot2, a dedicated visualization library, and dplyr, a data processing library can greatly reduce the data processing tasks required to diversify graphs

Figure 6-39 Visualization of gdpPercap and lifeExp in gapminder data by year

04 Exploring data using visualization

Visual Exploration of gapminder Data (3)

- Things we can know from the graph
  - In Europe, many countries recorded high gdpPercap and lifeExp early on.
  - Since the '70s, gdpPercap and lifeExp of many countries in Asia and the Americas have been growing rapidly.
  - Many African countries are staying at low gdpPercap and lifeExp.
  - As gdpPercap and lifeExp increase, the relationship between the two variables tends to gradually become linear.
  - During the observed period, gdpPercap and lifeExp in almost all countries rose overall (excluding some African countries).
  - The difference between gdpPercap minimum value and maximum value increased. In other words, the gap widened.

04 Exploring data using visualization

Changes and Characteristics of Economic Indicators in Kuwait (1)

- In 1952, we can find **one Asian country** with a very high gdpPercap.

```
> gapminder %>% filter(year == 1952 & gdpPercap > 10000 & continent == "Asia")
# A tibble: 1 x 6
  country continent  year lifeExp    pop gdpPercap
  <fct>   <fct>     <int>   <dbl>  <int>     <dbl>
1 Kuwait  Asia       1952    55.6 160000   108382.
```

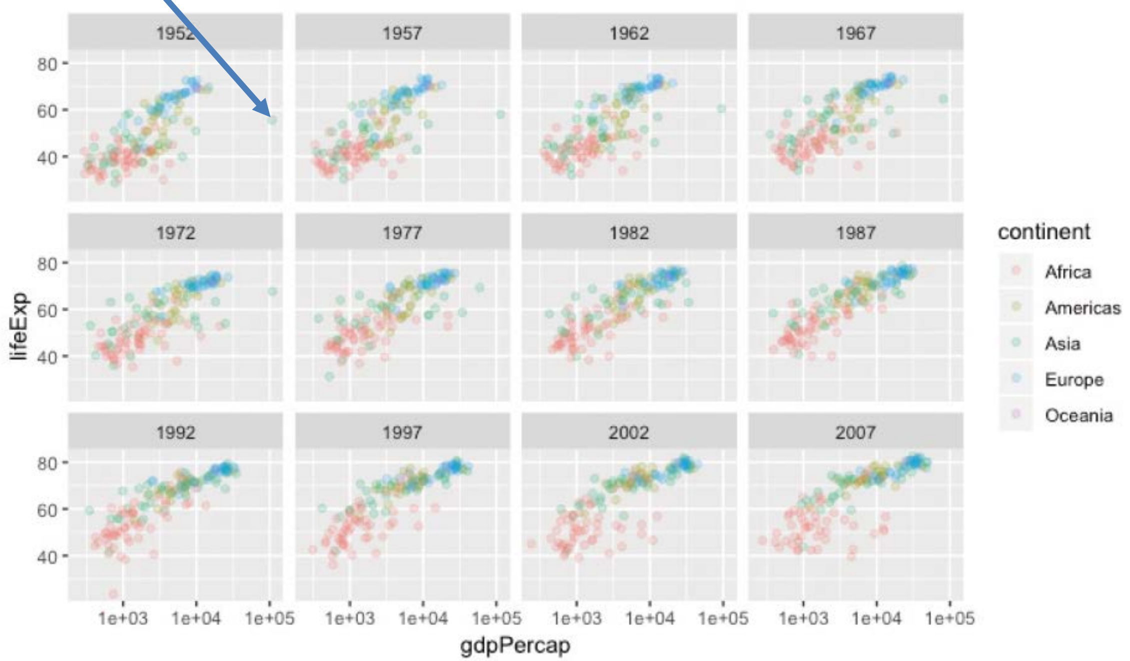


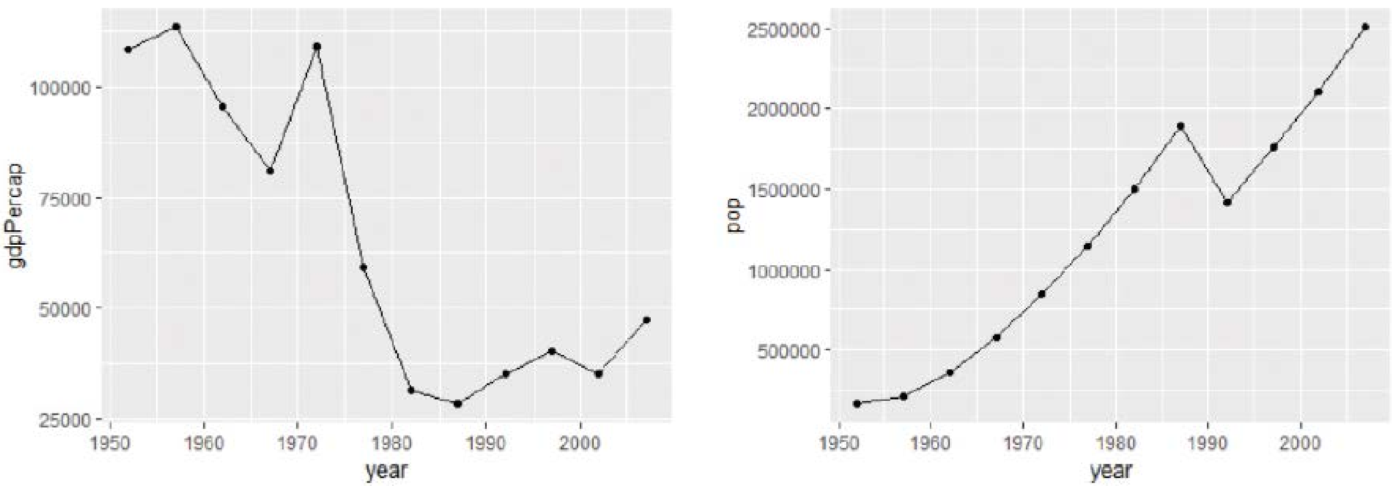
Figure 6-39 Visualization of gdpPercap and lifeExp in gapminder data by year

04 Exploring data using visualization

Changes and Characteristics of Economic Indicators in Kuwait (2)

- Let's visualize gdpPercap and pop changes after 1952.

```
> gapminder %>% filter(country == "Kuwait") %>% ggplot(aes(year, gdpPercap)) +
geom_point() + geom_line()                    # [Figure 6-40(a)]
> gapminder %>% filter(country == "Kuwait") %>% ggplot(aes(year, pop)) + geom_
point() + geom_line()                         # [Figure 6-40(b)]
```



(a) change of gdpPercap (b) change of pop

Figure 6-40 Graph of the changes gdpPercap and pop by year in Kuwait.

04 Exploring data using visualization

NOTE

Since the outbreak of the fourth Middle East War in October 1973, six oil-producing countries in the Persian Gulf have started to raise prices and reduce production, and the price of the crude oil notice, which stood at $2.9 a barrel, has surpassed $4 a barrel. In January 1974, it rose to 11.6 dollars, quadrupling in two to three months. Due to this situation, major advanced economies had to undergo stagflation accompanied by a higher increase in prices and negative growth in 1974.

NOTE

Kuwait gained independence from Britain in 1961 and was occupied by Iraq during the 1990-1991 Gulf War. Iraq forcibly incorporated Kuwait into its 19th state since Iraqi forces invaded Kuwait in 1990, but multinational forces ousted it in January 1991. In March of this year, Kuwait regained its territory.

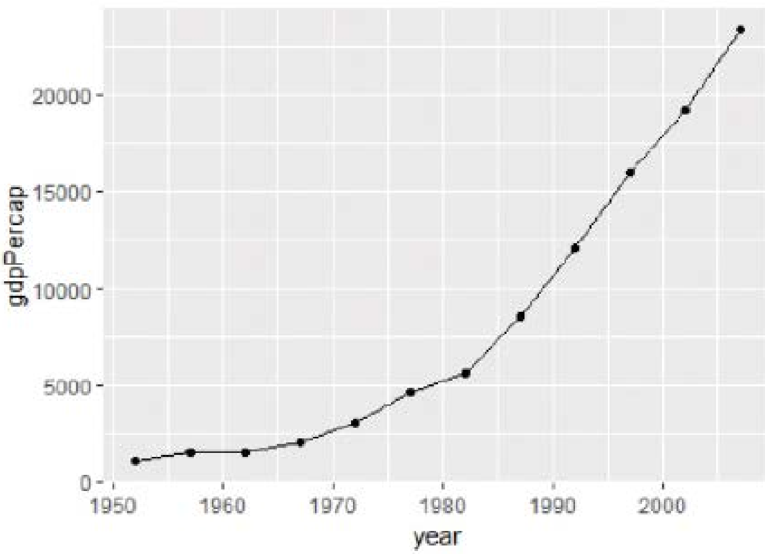
Kuwait's territory is narrow, but it has 10 percent of the world's oil reserves. Thanks to this, for a time it once recorded the world's No. 1 per capita GDP and is now the fourth richest country in the world. By strongly pushing for the nationalization of the oil industry, its oil fields contain most of Kuwait's capital, and Kuwait covers 90 percent of its national income with oil exports. It is a country that is get its national income with just oil.

04 Exploring data using visualization

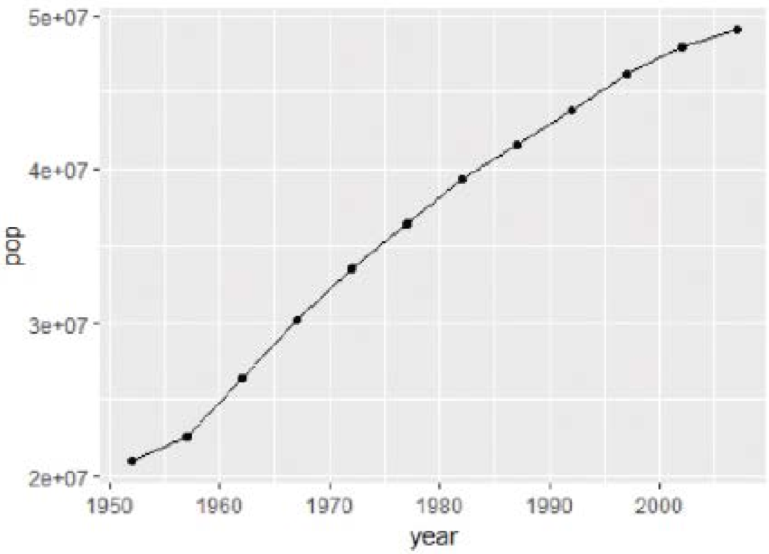
Changes and Characteristics of Economic Indicators in Kuwait (3)

- Let's compare it to Korea, the same Asian country.

```
> gapminder %>% filter(country == "Korea, Rep.") %>% ggplot(aes(year, gdpPercap))
+ geom_point() + geom_line()                        # [Figure 6-41(a)]
> gapminder %>% filter(country == "Korea, Rep.") %>% ggplot(aes(year, pop)) +
geom_point() + geom_line()                          # [Figure 6-41(b)]
```



(a) change of gdpPercap



(b) change of pop

Figure 6-41 Graph of the changes gdpPercap and pop by year in Korea.

04 Exploring data using visualization

Changes and Characteristics of Economic Indicators in Kuwait (4)

- In order to effectively observe changing gdpPercap and pop at the same time, it is also a good way to use GDP(gross domestic product) to check the economic size of the whole country.
- Use mutate function of the dplyr library to calculate GDP from gdpPercap and pop.

```
> gapminder %>% filter(country == "Kuwait" | country == "Korea, Rep.") %>% mutate(gdp =
gdpPercap*pop) %>% ggplot(aes(year, gdp, col = country)) + geom_point() + geom_line()
```

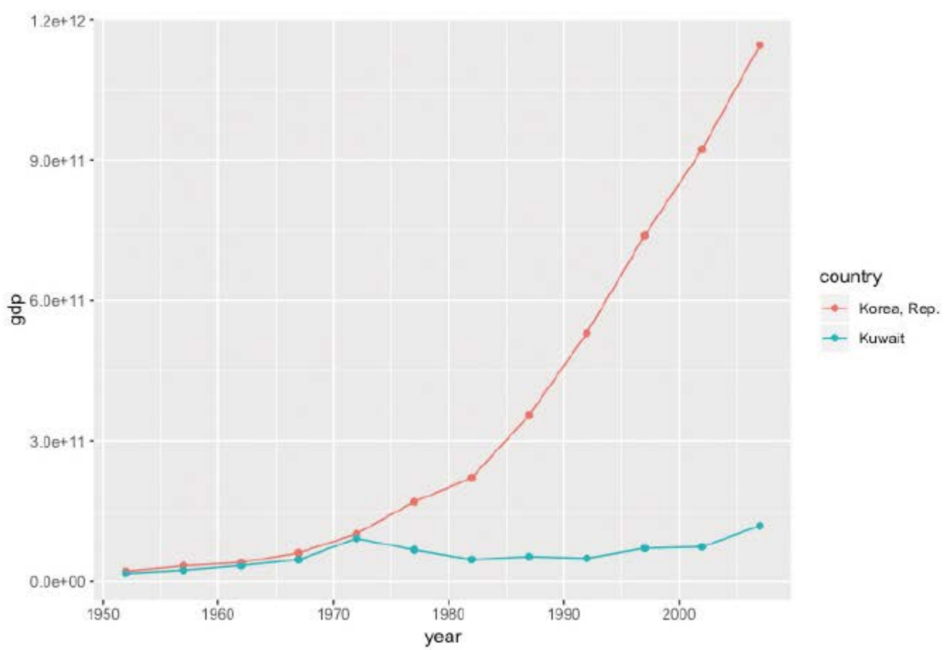


Figure 6-42 Graph of the changes GDP by year in Korea and Kuwait

04 Exploring data using visualization

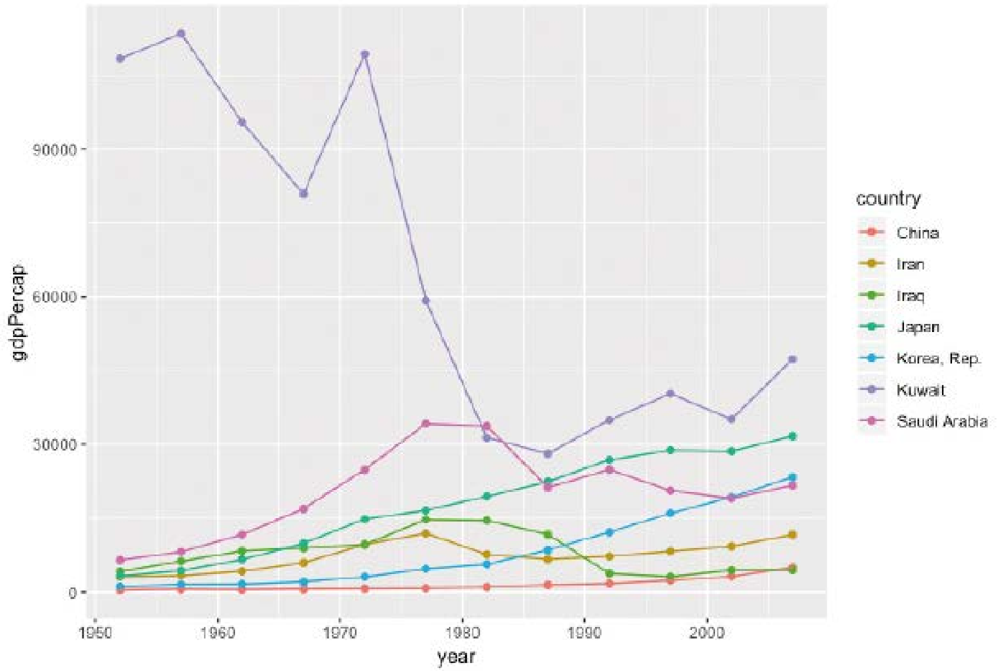
- Differences in Economic Indicator's change by Industrial Type (1)
  - Let's visualize and compare Asia's leading industrial nations(Korea, China, Japan) and oil-producing nations(Kuwait, Saudi Arabia, Iran, Iraq) together through gdpPercap, pop, gdp values.

04 Exploring data using visualization

■ Differences in Economic Indicator's change by Industrial Type (2)

```
# [Figure 6-43(a)] Comparison of changes in gdpPercap
> gapminder %>% filter(country == "Kuwait" | country == "Saudi Arabia" | country ==
"Iraq" | country == "Iran" | country == "Korea, Rep." | country == "China" | country ==
"Japan") %>% ggplot(aes(year, gdpPercap, col = country)) + geom_point() + geom_line()
```

(a) change of gdpPercap

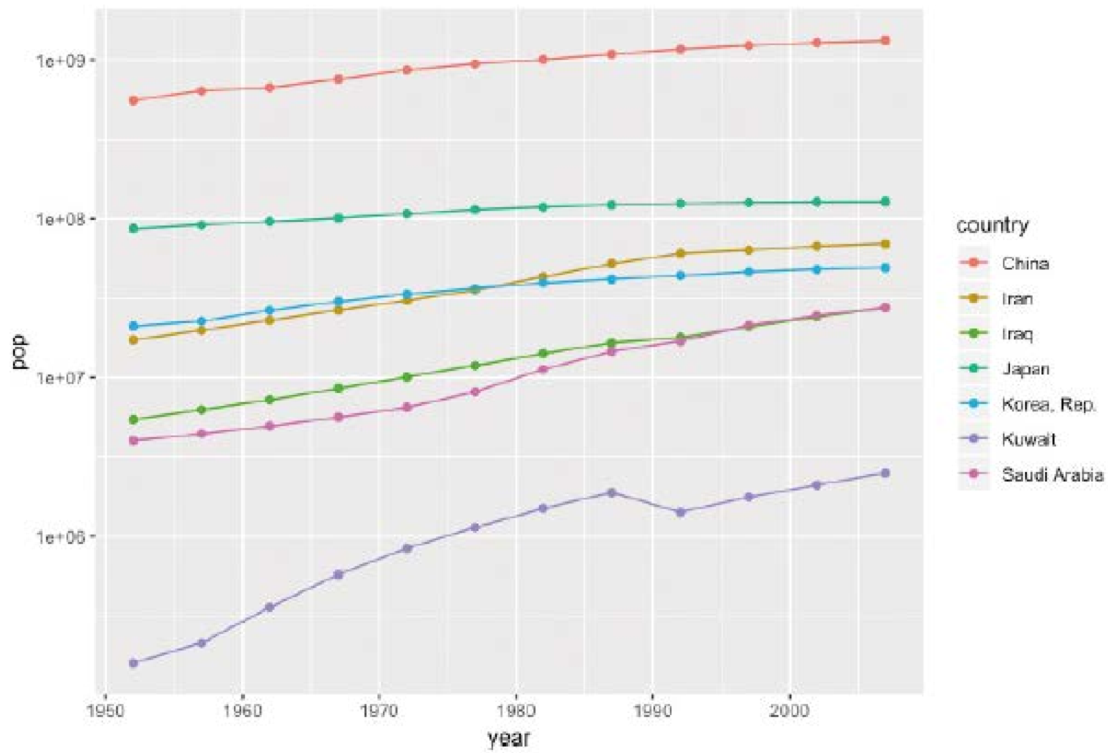


04 Exploring data using visualization

Differences in Economic Indicator's change by Industrial Type (3)

```
# [Figure 6-43(b)] Comparison of changes in pop
> gapminder %>% filter(country == "Kuwait" | country=="Saudi Arabia" | country ==
"Iraq" | country == "Iran" | country == "Korea, Rep." | country == "China" | country ==
"Japan") %>% ggplot(aes(year, pop, col=country)) + geom_point() + geom_line()
```

(b) change of pop



04 Exploring data using visualization

Differences in Economic Indicator's change by Industrial Type (4)

```
# [Figure 6-43(c)] Comparison of changes in gdp
> gapminder %>% filter(country == "Kuwait" | country == "Saudi Arabia" | country ==
"Iraq" | country == "Iran" | country == "Korea, Rep." | country == "China" | country ==
"Japan") %>% mutate(gdp = gdpPercap*pop) %>% ggplot(aes(year, gdp, col = country)) +
geom_point() + geom_line() + scale_y_log10()
```

(c) change of gdp

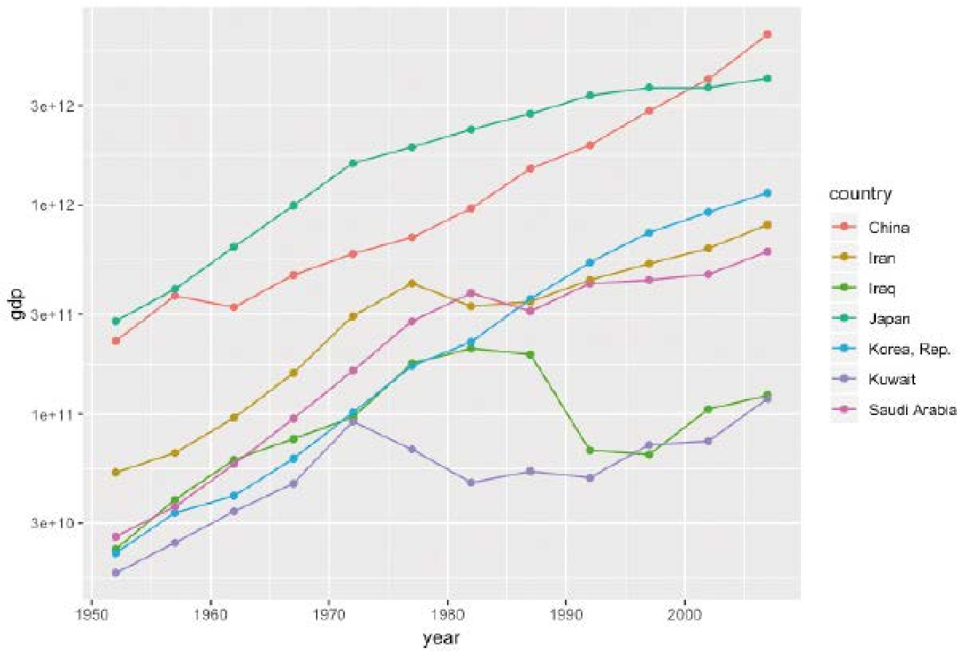


Figure 6-43 Graph of comparing changes the seven countries by year

04 Exploring data using visualization

Differences in Economic Indicator's change by Industrial Type (5)

- Visualizations of three items show the following facts:
  - The gdp of Middle Eastern countries suffered similar, wide increases/ decreases between 1970 and 1990, but while the gdp of South Korea, China and Japan continued to increase.
  - The population of Middle Eastern countries has increased rapidly since the corresponding period (1970~1990).
  - GdpPercap in countries in the Middle East showed a low growth rate compared to the corresponding period and the subsequent increase in gdp. It can be explained that gdpPercap shows a lower growth rate compared to gdp due to the high growth rate of pop since the corresponding period. In contrast, gdpPercap in South Korea, China and Japan are similar to the increase in gdp.