Data Science and R
5.1. Packages, Iterations, Conditionals

SUNG KYUN KWAN UNIVERSITY

Recap Data Frames

- Loading datasets from R library using `data()`
- Understanding basic information about a data frame using `str()`,`summary()`,`View()`, etc.
- Subset a data frame using `subset()`
- Neat coding using `with()`
- Conditional using `ifelse()`
- Creating new columns with aggregated information using all functions above

Installing Packages

- Packages are R functions and datasets that are stored in a library
- R comes with a standard set of packages
- Others can be downloaded and installed
- To see what packages are currently loaded in our environment
  - `search()`
- We are going to install and load the “MASS” package
  - `install.packages("MASS")`
  - `library(MASS)   # load package MASS into memory`
- Typing `data()` now should give an additional bunch of datasets in the `"MASS"` package

Iterations: `for()` Loops

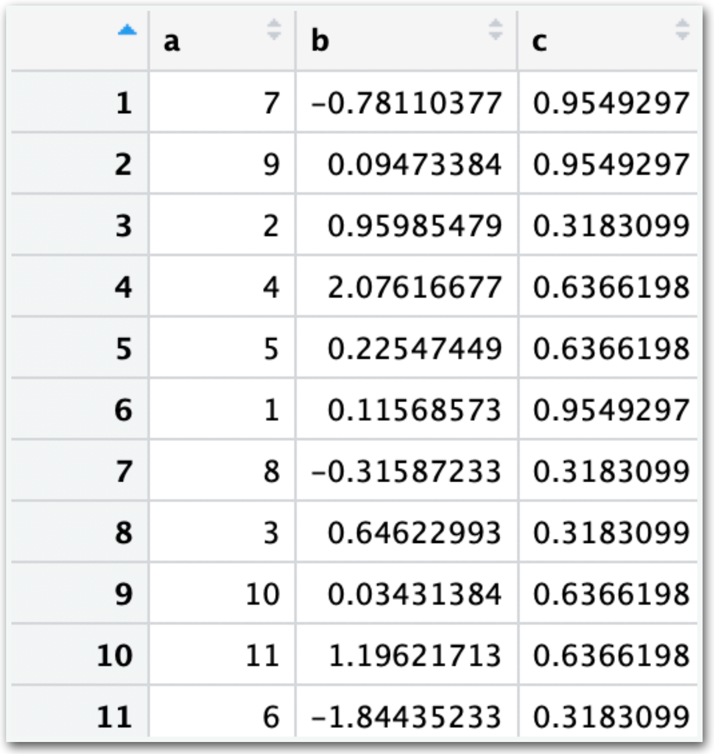
- Iterations or loops are used to execute a set of code repeatedly
- Usually used when you need to go through a vector, matrix, list or data frame and do a set of code to some or all of the elements in the data structure
- Need to specify a variable over a vector or range of values for the loop to be executed
- Example: multiply each element in the vector by two and print it out

```
# 1. Access elements directly
myvec <- c(2,3,5,7,11,13)
for (num in myvec) { # iterate over all elements in a
    print(num*2)
}
# Access elements using index
for (i in 1:length(myvec)) {
  print(myvec[i]*2)
}
```

Looping over Data Frames

- Getting the medians of all columns

```
df <- data.frame(
    a = sample(1:11, 11),
    b = rnorm(11),
    c = sample(1:3,11,replace=T)/pi)
for (i in 1:ncol(df)) { # loop by index
     print(median(df[[i]])) # print median of current column
}
[1] 6
[1] 0.1156857
[1] 0.6366198
```



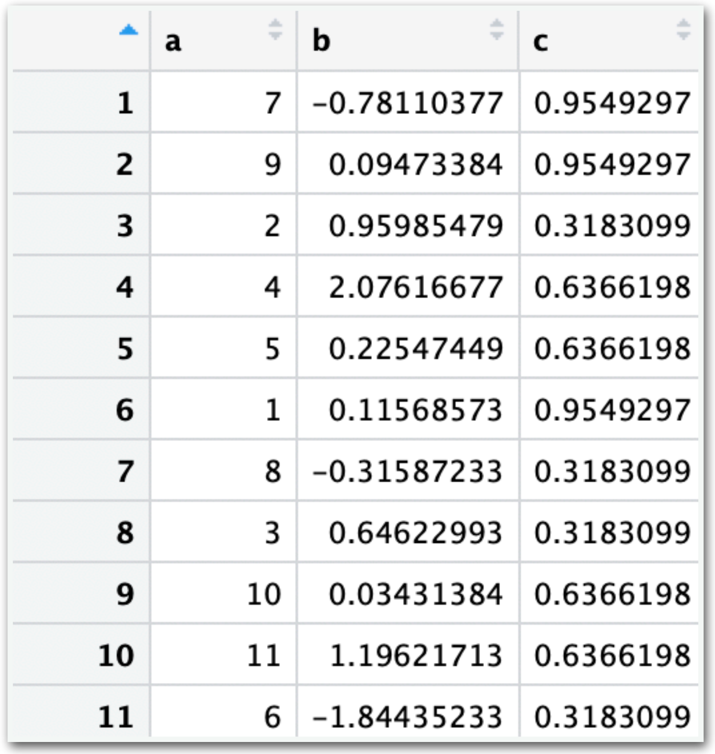
	a	b	c
1	7	-0.78110377	0.9549297
2	9	0.09473384	0.9549297
3	2	0.95985479	0.3183099
4	4	2.07616677	0.6366198
5	5	0.22547449	0.6366198
6	1	0.11568573	0.9549297
7	8	-0.31587233	0.3183099
8	3	0.64622993	0.3183099
9	10	0.03431384	0.6366198
10	11	1.19621713	0.6366198
11	6	-1.84435233	0.3183099

Conditional: if…else

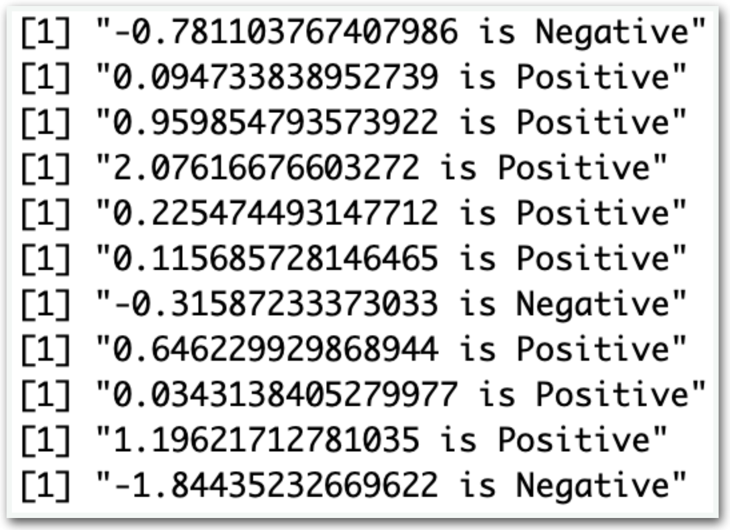
```
if (test_expression1)
{
     <statement1>
} else  if  (test_expression2) {
     <statement2>
} else {
     <statement3>
}
```

- Determine if each number in column b of df is positive, negative or zero

```
for (num in df$b) { # loop through numbers in column "b"
  if (num < 0) {
    num_status <- "Negative"
  } else if (num > 0) {
    num_status <- "Positive"
  } else { # neither negative nor positive
    num_status <- "Zero"
  }
 cat(num, "is", num_status, "\n")
}
```



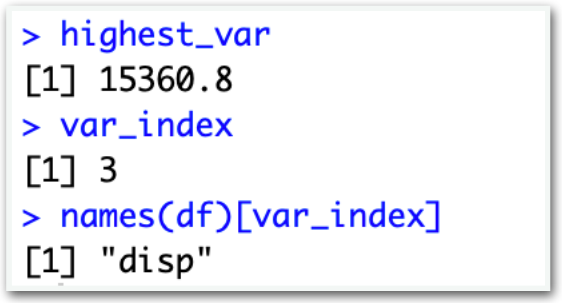
	a	b	c
1	7	-0.78110377	0.9549297
2	9	0.09473384	0.9549297
3	2	0.95985479	0.3183099
4	4	2.07616677	0.6366198
5	5	0.22547449	0.6366198
6	1	0.11568573	0.9549297
7	8	-0.31587233	0.3183099
8	3	0.64622993	0.3183099
9	10	0.03431384	0.6366198
10	11	1.19621713	0.6366198
11	6	-1.84435233	0.3183099



```
[1] "-0.781103767407986 is Negative"
[1] "0.094733838952739 is Positive"
[1] "0.959854793573922 is Positive"
[1] "2.07616676603272 is Positive"
[1] "0.225474493147712 is Positive"
[1] "0.115685728146465 is Positive"
[1] "-0.31587233373033 is Negative"
[1] "0.646229929868944 is Positive"
[1] "0.0343138405279977 is Positive"
[1] "1.19621712781035 is Positive"
[1] "-1.84435232669622 is Negative"
```

Which Column has Highest Variance?

```
df <- na.omit(mtcars) # remove missing values, known as "Not Available" NA values from data frame
highest_var <- 0.0 # initialise highest variance to 0
var_index <- 0 # initialise highest index to 0
for (i in 1:ncol(df)) # loop over all columns
{
      if(is.numeric(df[[i]])) # Compute variance for numeric columns only
      {
         current_var = var(df[[i]])
        if(current_var > highest_var) # if variance of this col > highest variance so far
        {
          highest_var <- current_var # set the highest variance to be this column’s variance
          var_index <- i # set the index of highest variance to be this column index
        } # end if
      } # end if
} # end for
cat("Highest variance:", highest_var) # print the highest variance
var_index # print the index of the column with the highest variance
cat("Column:", names(df)[var_index]) # print column name with highest variance
```



```
> highest_var
[1] 15360.8
> var_index
[1] 3
> names(df)[var_index]
[1] "disp"
```

Exit loop: `break`

- Sometimes you want the loop to exit when certain conditions have been met, so `break` will tell the R interpreter to pass control to the instruction immediately after the end of the loop (if any)

```
# Check to see if the letter "x" exists in a random string of characters
taken from the alphabet

my_letter <- "x"

random_word = sample(letters, 13)

for (letter in random_word){

  print(letter)

  if (letter == my_letter){

    print(letter, "FOUND")

    break  # exit the loop when a match is found

  }

}

print(random_word)
```

Skip and Continue with loop: `next`

- To discontinue a particular iteration and jump to the next cycle, use `next`
- In fact, it jumps to the evaluation of the condition holding the current loop

```
# Print odd numbers between 1 and 20
for (k in 1:20){
  if (!k %% 2){ # same as k %% 2==0, i.e. if k is divisible by 2
    next # skip the rest of the loop; go back to top of loop with the next k
  }
  print(k) # only k's that are not divisible by 2 get here
} # end for
```

Tips

- Install new packages using, e.g. package ‘MASS’:
  - `install.packages('MASS')`
  - `library(MASS) # can use data and functions from some_package directly`
  - `data(Animals) # load the dataset 'Animals'`
- Use `na.omit()` to remove missing (NA) values from a dataset
- Other loop constructs are `while()` and `repeat()`
- Interrupt (exit) the loop using `break`
- Ignore remaining code in loop and go back to evaluation of the condition using `next`