STAT 302: Statistical Computing	Winter 2024

Lecture 7: Midterm Review

Yikun Zhang   *February 13, 2024*

This note is designed to give a brief review on lecture materials [Zhang, 2023] and highlight those important concepts/results in STAT 302 before the midterm exam. The review is by no means comprehensive and in order to excel at the midterm exam, a student is expected to master those R fundamentals in the course instead of simply memorizing the key functions or commands in R.

Disclaimer: Parts of this note are modified from the midterm review of STAT 133 in Fall 2016 at UC Berkeley by Professor Deborah Nolan.

1 R Basics and Syntax

- **Binary arithmetic operators:** +, -, *, /, %% (for mod), %/% (for integer division), and ^ (exponentiation), etc.

- **Comparison operators:** >, <, <=, >=, ==, and !=. The returning value of a comparison statement is TRUE or FALSE.

- **Logical operators:** &, |, and !. A logical operator takes one or more comparison statements and return TRUE or FALSE.

⋆ Notes: Pay attention to the order/priority of operations as well as the difference between simple and compound expressions.

Data Types in R:

- Basic types in R: numeric (and integer), character, logical, factor, etc.

- Factor has a level (string) and is stored internally as an integer.

- Special values include NA (not available), NaN (not a number), NULL (empty vector), and Inf (infinity).

We can use the built-in function class() to determine the data type of an R object.

⋆ Note: The function typeof() determines the (R internal) type or storage mode of any R object. Its returning value may not be the same as class().

2 Data Structures in R

Vector calculations:

- Vector is an ordered container of primitive elements of the same data type.

- R is vectorized so that we can perform operations on vectors.

- Remember the recycling rule in R when the vectors are of different lengths:

```
> x = rep(c(1,2), times = c(2, 4))
> y = 1:3
> x^y
[1] 1 1 8 2 4 8
```

Matrices and arrays: A matrix or an array is a rectangular collections of primitive elements of the same data type. They are stored in R as a vector with shape information.

Lists: A list is an ordered container of objects, e.g., mixtures of vectors, data frames, lists, functions.

Data frames: A data frame is an ordered container of vectors, where the vectors are of the same length and possibly different data types.

Practice Problem: Describe two important differences between a data frame and a matrix in R.

Answer: Here are some acceptable answers.

- A data frame is essentially a list of vectors of the same length, whereas a matrix is essentially a vector with shape information.

- Data frames can have columns/vectors that are different types, whereas all values in a matrix must be the same primitive element.

- Data frames can be indexed with $.

- Data frames require variable names.

Subsetting: We use the `family.txt` data as an example.

```
> head(family)
  firstName sex age height weight     bmi overWt
1       Tom   m  77     70    175 25.16239   TRUE
2      Maya   f  33     64    124 21.50106  FALSE
3       Joe   m  79     73    185 24.45884  FALSE
4    Robert   m  47     67    156 24.48414  FALSE
5       Sue   f  27     61     98 18.51492  FALSE
6       Liz   f  33     68    190 28.94981   TRUE

> sapply(family, class)
  firstName         sex         age      height      weight         bmi     overWt
"character" "character"   "integer"   "integer"   "integer"   "numeric"  "logical"
```

- Position – using the indices of elements, rows, columns that we want.

  ```
  > family[,1]
   [1] "Tom"    "Maya"   "Joe"    "Robert" "Sue"    "Liz"    "Jon"    "Sally"  "Tim"
  [10] "Tom"    "Ann"    "Dan"    "Art"    "Zoe"

  > family[c(1, 3, 6), ]
    firstName sex age height weight      bmi overWt
  1       Tom   m  77     70    175 25.16239   TRUE
  3       Joe   m  79     73    185 24.45884  FALSE
  6       Liz   f  33     68    190 28.94981   TRUE
  ```

- Exclusion – using the indices of elements to exclude.

```
> family[-(2:12), ]
  firstName sex age height weight      bmi overWt
1       Tom   m  77     70      175 25.16239   TRUE
13      Art   m  46     66      150 24.26126  FALSE
14      Zoe   f  48     62      125 22.91060  FALSE
```

- Logical – using a logical vector of the same length as the vector/rows/columns being subsetted. It will keep the elements corresponding to `TRUE`.

```
> family[(family$overWt) & (family$age > 60), ]
  firstName sex age height weight      bmi overWt
1       Tom   m  77     70      175 25.16239   TRUE
7       Jon   m  67     68      185 28.18797   TRUE
```

- Name – using a character vector of names of elements/rows/columns to keep.

```
> family[, "age"]
 [1] 77 33 79 47 27 33 67 52 59 27 55 24 46 48
```

⋆ Note: Subsetting a list or data frame via a double-square bracket returns the element or column within the list or data frame, but not a list or data frame of one element.

```
> class(family[["age"]])
[1] "integer"
> class(family["age"])
[1] "data.frame"
```

Practice Problem: Write down what the value of x will contain after each line of `R` code, if the commands are executed sequentially.

```
> x = seq(0, 8, length = 5)
> x
[1] 0 2 4 6 8

> x[x<4] = NA
> x
[1] NA NA 4 6 8

> x[5] = 10
> x
[1] NA NA 4 6 10

> x[] = 0
> x
[1] 0 0 0 0 0

> x = 12
> x
[1] 12
```

Practice Problem: Data on 37 parents of babies born at Kaiser Hospital in the 1960s is available in a data frame called `parents`. The variables age, ed, ht, and wt are the mother's age, education level, height and weight. The variables that start with the letter d are corresponding variables for the fathers.

```
> head(parents)
  age ed           ht wt dage ded             dht dwt marital inc
1 27 College       62 100 31 College           65 110 Married [2500, 5000)
2 33 College       64 135 38 College           70 148 Married [7000, 8000)
3 28 High School   64 115 32 Some High School NA NA Married  [5000, 6000)
4 36 College       69 190 43 Some College      68 197 Married [12500, 15000)
5 23 College       67 125 24 College           NA NA Married  [2500, 5000)
6 25 High School   62 93  28 High School       64 130 Married [7000, 8000)
```

1. Write an R expression to find the subset of parents where the mother is over 40.

```
parents[parents$age > 40, ]
```

2. Write an R expression using an apply function to return the class of each variable in the data frame.

```
sapply(parents, class)
```

3. Write one R expression using an apply function to return the number of NAs in each variable (recall that there is a function called `is.na()` that returns a logical indicating the presence of NAs).

```
sapply(parents, function(x){
  sum(is.na(x))
})
```

Practice Problem: Here is a list in R:

```
> x
$a
[1] 0.03895442 0.77658866 0.83532332

$b
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Write one line of R code to extract the first row of the matrix.

```
x$b[1, ]
```

```
x[["b"]][1, ]
```

⋆ Notes: You should know how to use `dim()`, `head()`, `class()`, `length()` functions with the data structures in R. In addition, you should also be able to use `sapply()`, `lapply()`, `apply()`, `tapply()`, and `mapply()` functions with these structures.

Practice Problem: Suppose we have a matrix `m`, and we have just executed the following:

```
> dim(m)
[1] 500   3
```

```
> head(m)
          [,1]         [,2]      [,3]
[1,] 0.9231213  0.2142767 2.544545
[2,] 0.5669828  0.6981698 1.336078
[3,] 0.7077386  0.1447157 1.750822
[4,] 0.6723225  0.6040853 0.042359
[5,] 0.4925304 -0.4389639 1.504622
[6,] 0.8370886  0.3683489 1.456362
```

We need to create a vector containing the sum of the squared entries in each row of `m`. Use an anonymous function and the `apply()` function to do this:

```
apply(m, 1, function(x) {
  sum(x^2)
})
```

3 Control Flow and Basic Iterations

Conditional evaluation: `if` and `if-else` statements.

```
if (condition to evaluate to TRUE or FALSE) {
  code block to be run if above is TRUE
}
```

and

```
if (condition to evaluate to TRUE or FALSE) {
  code block to be run if above is TRUE
} else {
  code block to be run if above is FALSE
}
```

Iterations: Looping with `for` and `while` statements.

```
> x = c(10, 2, 6, 5)
> for (i in 1:3) {
    print(x[i])
  }
[1] 10
[1] 2
[1] 6
```

4 Data Manipulation via `tidyverse`

Pipes %>% operator:

- x `%>%` f `%>%` g is equivalent to $g(f(x))$.

- x `%>%` $f(y, .)$ is equivalent to $f(y, x)$.

`dplyr` functions: Some of the most important `dplyr` verbs (functions):

- `filter()`: subset rows based on a condition.

- `group_by()`: define groups of rows according to a column or specific condition.

- `summarize()`: apply computations across groups of rows.

- `arrange()`: order rows by value of a column.

- `select()`: pick out given columns.

- `mutate()`: create new columns.

- `mutate_at()`: apply a function to given columns.

tidyr functions: Two of the most important `tidyr` verbs (functions):

- `pivot_longer()`: make "wide" data longer.

- `pivot_wider()`: make "long" data wider.

5 Graphics in R

Data type and plot correspondence:

- One variable:
  factor: barplot, dot chart, line plot, etc.
  numeric: rug plot, density plot, histogram, boxplot, etc.

- Two variables:
  Two factors: side-by-side bar plot, superposed line plots, mosaic plot, side-by-side dot chart, etc.
  Two numerics: scatter plot, line plot if one variable is time numeric + factor: super-posed density curves, side-by-side boxplots or violin plots, line plots, etc.

- More details can be found in the ggplot2 cheet sheet and the reference book.

Plotting principles:

- Make the data standout: symbols, over plotting, interference, jittering, scale, etc.

- Facilitate comparison: emphasize the important difference; don't jiggle the baseline; juxtapose, superpose and choice of scale; banking to 45 degrees; perception – color, shape.

- Information rich: color; reference lines and markers; legends and labels; captions.

ggplot essentials:

- `ggplot()` initiates a plot. We can provide a data frame and a mapping between the variables in the data frame and plot aesthetics.

- Layers - we add geoms/glyphs to the plot with `geom_XXX()` functions. Examples include
  `geom_point()` for points in a scatter plot,
  `geom_line()` for connecting dots to make line plots,

geom_smooth() for averaging the y-values with similar x-values to create a smooth curve as a function of x,

geom_bar() for making bars in a bar plot,

geom_boxplot() for making boxes in a box plot,

geom_density() for computing and drawing kernel density estimate, which is a smoothed version of the histogram,

geom_histogram() for making bars in a histogram.

In the layer, we can also provide the data frame and the mapping of the variables to aesthetics.

- Scales - these are associated with the aesthetics. Each aesthetic has a scale. We use the scale_XX_xxxx() functions to specify details. The first argument in each scale function is name, which provides the axis or legend name. A short cut function called labs() allows use to specify the name to all of our aesthetics without having to call the scale functions, e.g., labs(x = "my x", y = "my y", color = "color legend title").

- Theme - themes are used for general details, such as font type and size, background color, etc.

6   Writing Functions in R

Function signatures:

- Required parameters – have no default value provided, usually come first in the signature.

- Optional parameters – have default values (which may be computed from other parameters).

- Function calls – R sets up the call frame with default values for parameters, then values are assigned for the arguments mentioned by name in the call. R will first look for the value of a variable by name in the function environment. If it can't be found, then it will resort to its parent environment, which is the global environment in this case, etc.

- <<- is used to assign values to variables in the parent environment.

- Lazy evaluation – If an expression is provided for a parameter value, then this expression is not evaluated until the variable is referenced in the computations for the function call.

- Debugging tools and testing – traceback(), browser(), debug(), etc.

Practice Problem: Write a function to compute the sum of the absolute deviations from the median for an input vector. For example, for a vector x = 1:3, x has a median of 2, and the absolute deviations from the median are 1, 0, and 1 so the sum of the absolute deviations from the median is 2.

Call the function sadm(). This function has one parameters: x, which is required and holds the numeric vector that will be operated on.

```
sadm = function(x){
  res = sum(abs(x - median(x)))
  return(res)
}
```

Practice Problem: Update the function sadm() to handle NA values. Add a parameter: na.rm, which determines whether NAs are to be removed from the computation. The na.rm parameter has a default value of FALSE.

```
sadm = function(x, na.rm = FALSE){
  if (na.rm) {
        x = x[!is.na(x)]
  }
   res = sum(abs(x - median(x)))
   return(res)
}
```

7   Simulation Basics

- Simple random selection with `sample()`

- Replicating the experiment with `replicate()`.

Practice Problem: Someone wants to study the distribution of the sum of three rolls of a die. To do this, he/she designs a simulation study. In the first step, he/she writes a function called `sum3()` to generate the sum of three random tosses of a fair die. In the second step, he/she uses this function to generate 1,000 of these sums.

1. Write the function for the first step.

```
sum3 = function(){
  res = sum(sample(1:6, 3, replace = TRUE))
  return(res)
}
```

2. Write one line of code that uses the function from the first step to generate the 1,000 random sums.

```
replicate(1000, sum3())
```

References

Y. Zhang. STAT 302 Lecture Slides, 2023. URL https://zhangyk8.github.io/teaching/stat302_uw.