Stat 133 HW03: Flow Control Structures and Functions with R

Hye Soo Choi and 23274190

Introduction

This assignment has two purposes:

a) to familiarize you with control flow structures in R
b) to introduce you to writing functions in R

Submit your assignment to bcourses, specifically turn in your **Rmd** (R markdown) file as well as the produced pdf file. Make sure to change the argument `eval=TRUE` inside every testing code chunk.

Last Element

Write a function `last()` that takes a vector (or factor) and returns the last element in the vector.

```
# write your function
# last()
last <- function(vec){
  return (vec[length(vec)])
}
```

Test it:

```
last(c('A', 'E', 'I', 'O', 'U'))
```

```
## [1] "U"
```

```
last(c(2, 4, 6, 8, 10))
```

```
## [1] 10
```

If-then-else

Write a function `multfive()` that takes a number and determines whether the number is multiple of 5. If the provided number is multiple of five, then the output must be: `it is multiple of five`. Conversely, if the provided number is not a multiple of five, then the output must be: `it’s not a multiple of 5`.

```r
# write your function
# multfive()
multfive <- function(num){
  if(num %% 5 == 0){
  return ('it is multiple of five')
  }
  else{
   return("it's not a multiple of 5")
  }
}
```

Test it:

```r
# multiple of five
multfive(10)
```

```
## [1] "it is multiple of five"
```

```r
# not a multiple of five
multfive(33)
```

```
## [1] "it's not a multiple of 5"
```

Create your histogram plotting function

Write a function `histogram()` that plots a histogram with added vertical lines for the following summary statistics: minimum value, median, mean, and maximum value. The main idea is to wrap the high-level function `hist()` and then plot the lines with a low-level plotting function.

Define your function with the following requirements:

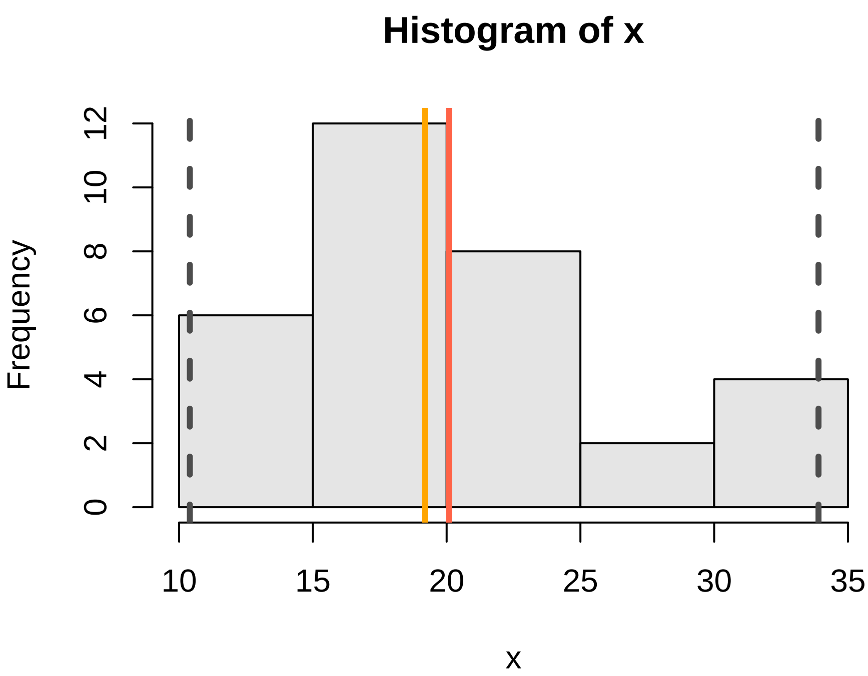
- bars of histogram colored in "gray90""
- line of minimum value in color "gray30", and dashed type
- line of maximum value in color "gray30", and dashed type
- line of median value in color "orange"
- line of mean value in color "tomato"
- all lines (min, max, median, mean) with a width of 3

```r
# write your function
# histogram
histogram <- function(data){
  hist(data, col = 'gray90', xlab = 'x', ylab ='Frequency',main='Histogram of x')
  abline(v =c(min(data), max(data), median(data), mean(data)),
         col= c('gray30','gray30', 'orange','tomato'),lty = c(2,2,1,1), lwd=3)

}
```

Test it:

```r
histogram(mtcars$mpg)
```



Converting Fahrenheit Degrees

The table below shows the different formulas for converting Fahrenheit degrees into other scales:

Units	from Fahrenheit
Celsius	(°F - 32) x 5/9
Kelvin	(°F + 459.67) x 5/9
Reaumur	(°F - 32) x 4/9
Rankine	°F + 459.67

Write a function that converts from Fahrenheit degrees into each type of the four alternative scales. This implies writing four different functions:

- `to_celsius()`
- `to_kelvin()`
- `to_reaumur()`
- `to_rankine()`

```r
# write your functions
to_celsius <- function(faren){
  return ((faren - 32) * 5 / 9)
}
```

```r
to_kelvin <- function(faren){
  return((faren + 459.67) * 5 / 9)
}
to_reaumur <- function(faren){
  return((faren - 32) * 4 / 9)
}
to_rankine <- function(faren){
  return(faren + 459.67)
}
```

Test them:

```r
to_celsius(34)
```

```
## [1] 1.111111
```

```r
to_kelvin(34)
```

```
## [1] 274.2611
```

```r
to_reaumur(34)
```

```
## [1] 0.8888889
```

```r
to_rankine(34)
```

```
## [1] 493.67
```

Using switch()

Create a function `convert()` that converts Fahrenheit degrees into the specified scale. Use `switch()` and the previously defined functions—`to_celsius()`, `to_kelvin()`, `to_reaumur()` and `to_rankine()`—to define `convert()`. Use two arguments: `x` and `to`

```r
# write your function
# convert()
convert <- function( x, to ){
  scaled_temp <- switch(
    to,
    celsius = to_celsius(x),
    kelvin = to_kelvin(x),
    reaumur = to_reaumur(x),
    rankine = to_rankine(x)
  )
  return(scaled_temp)
}
```

Test it:

```r
convert(32, "celsius")
```

```
## [1] 0
```

```r
convert(32, "kelvin")
```

```
## [1] 273.15
```

```r
convert(32, "reaumur")
```

```
## [1] 0
```

```r
convert(32, "rankine")
```

```
## [1] 491.67
```

Permutations

Write a function `permute()` that calculates the number of permutations of k objects from a set of n objects.

```r
# write your function
# permute()

permute <- function(n,k){
  if (n < 0 | k < 0){
    print('stop: both inputs should be non-negative numbers')
    # I wanted this to be stop('both inputs should be non-negative numbers')
    # but when evaluating permute(-6, 6), this function stops and prevent pdf from being produced.
  }
  else if(n < k){
    return ( 0 )
  }
  else{
    n <- as.integer(n)
    k <- as.integer(k)
    return( factorial(n)/ factorial(n-k))
  }
}
```

Test it:

```r
permute(6, 2)
```

```
## [1] 30
```

Make sure that the function checks that both n are k are non-negative numbers (if any of them is negative, the function must stop). Also make sure that if n is less than k, the result is zero. In addition, n and k should be coerced as integers.

```r
# the following calls should not work
permute(2, 6)
```

```
## [1] 0
```

```r
permute(-6, 6)
```

```
## [1] "stop: both inputs should be non-negative numbers"
```

Average function with for loop

Create a function `average()` using a ***for loop*** to compute the mean. `average()` takes a numeric vector and returns the average.

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

```r
# write your function
# average()
average <- function(num_vec){
  total <- 0
  for(i in 1:length(num_vec)){
    total <- total + num_vec[i]
  }
  return ( total / length(num_vec))
}
```

Test it:

```r
average(1:5)
```

```
## [1] 3
```

```r
mean(1:5)
```

```
## [1] 3
```

Geometric Mean function

The formula of the geometric mean is:

$$\left(\prod_{i=1}^{n} x_i \right)^{1/n}$$

Write a function `geomean()` that computes the geometric mean of a vector of positive numbers, using a ***for loop***:

```r
# write your function
# geomean()
  geomean <- function(num_vec){
    product <- 1
    for(i in 1:length(num_vec)){
      product <- product * num_vec[i]
    }
    return (product ^ (1/length(num_vec)))
  }
```

Test it:

```r
geomean(1)
```

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

```r
geomean(1:5)
```

```
## [1] 2.605171
```

Frequency Table

Write a function `freq_table()` that takes a factor and generates a frequency table with 5 columns:

1) `category`: the levels of the factor
2) `count`: absolute frequency
3) `prop`: relative frequency (use four decimal places)
4) `cumcount`: cumulative absolute frequency
5) `cumprop`: cumulative relative frequency (use four decimal places)

Make sure that the input is a factor (otherwise the function should stop). Likewise, the output should be in `data.frame` form.

```r
# write your function
# freq_table()

freq_table <- function( data){
  if(is.factor(data) == FALSE){
    stop ('input should be a factor')
  }
  else{
  category <- levels(data)
  count <- as.vector(table(data))
  prop <- round(count/ length(data), digits = 4)
  cumcount <- cumsum(count)
  cumprop <- round (cumcount / length(data), digits = 4)
  }
  freq_table <- data.frame(category = category, count = count, prop = prop,
```

```
                           cumcount = cumcount, cumprop = cumprop)
  return (freq_table)
}
```

Test it:

```
# some factor
set.seed(13)
sizes <- factor(
  sample(c('small', 'medium', 'large'), size = 90, replace = TRUE)
)

# frequency table
freq_table(sizes)
```

```
##   category count   prop cumcount cumprop
## 1    large    23 0.2556       23  0.2556
## 2   medium    40 0.4444       63  0.7000
## 3    small    27 0.3000       90  1.0000
```

Summary Statistics Table

Write a function `stats()` that takes a numeric vector and generates the following descriptive statistics:

- `min`: minimum value
- `max`: maximum value
- `range`: range (max - min)
- `q1`: first quartile
- `q3`: third quartile
- `iqr`: inter-quartile range (q3 - q1)
- `median`: median
- `mean`: mean
- `sd`: standard deviation
- `NAs`: number of missing values `NA`

The function `stats()` should include an argument `na.rm` —that takes a logical value— so it can handle potential missing values. The output must be a data.frame of one column.

```
# write your function
# stats()
stats <- function (num_vec , na.rm = FALSE){
  min <- min(num_vec, na.rm = na.rm)
  max <- max(num_vec, na.rm = na.rm)
  range <- max - min
  q1 <- quantile(num_vec, na.rm = na.rm)[2]
  q3 <- quantile(num_vec, na.rm = na.rm)[4]
  iqr <- q3 - q1
  median <- median(num_vec, na.rm = na.rm)
```

```r
  mean <- mean(num_vec, na.rm = na.rm)
  sd <- sd(num_vec, na.rm = na.rm)
  NAs <- sum(is.na(num_vec))

  stats <- c( min, max,  range,  q1, q3,  iqr,  median,
             mean,  sd,  NAs)
  stats <- data.frame(stats)
  row.names(stats) <- c('min', 'max' , 'range', 'q1', 'q3' ,'iqr', 'median','mean', 'sd', 'NAs' )
  return( stats )
}
```

Test it:

```r
# no missing values
stats(1:10)
```

```
##           stats
## min     1.00000
## max    10.00000
## range   9.00000
## q1      3.25000
## q3      7.75000
## iqr     4.50000
## median  5.50000
## mean    5.50000
## sd      3.02765
## NAs     0.00000
```

```r
# missing values
stats(c(1:4, NA, 6:9, NA), na.rm = TRUE)
```

```
##         stats
## min    1.0000
## max    9.0000
## range  8.0000
## q1     2.7500
## q3     7.2500
## iqr    4.5000
## median 5.0000
## mean   5.0000
## sd     2.9277
## NAs    2.0000
```

Frequency Table and Summary Statistics

Having created the functions `freq_table()` and `stats()`, use them to write a function `univarite()` for producing summary statistics depending on the type of input. If the provided input is a numeric vector, then `stats()` should be called. In turn, if the provided input is a factor, then `freq_table()` should be called. If the input is not a numeric vector or a factor, then `univariate()` will print: `"x must be either a numeric vector or a factor"`

```r
# write your function
# univariate()
univariate <- function(data){
  if(is.numeric(data)){
    return (stats(data))
  }
  else if(is.factor(data)){
    return (freq_table(data))
  }
  else{
    print ('x must be either a numeric vector or a factor')
  }
}
```

Test it:

```r
# factor input
univariate(sizes)
```

```
##   category count   prop cumcount cumprop
## 1    large    23 0.2556       23  0.2556
## 2   medium    40 0.4444       63  0.7000
## 3    small    27 0.3000       90  1.0000
```

```r
# numeric input
univariate(1:10)
```

```
##           stats
## min     1.00000
## max    10.00000
## range   9.00000
## q1      3.25000
## q3      7.75000
## iqr     4.50000
## median  5.50000
## mean    5.50000
## sd      3.02765
## NAs     0.00000
```

This should not work:

```r
# this should cause an error
univariate(colors()[1:5])
```

```
## [1] "x must be either a numeric vector or a factor"
```