Stat 133 HW03: Flow Control Structures and Functions with R

Gaston Sanchez

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. For instance:

```
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`. For example:

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

```
## [1] "it is multiple of 5"
```

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

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

Create your histogram plotting function

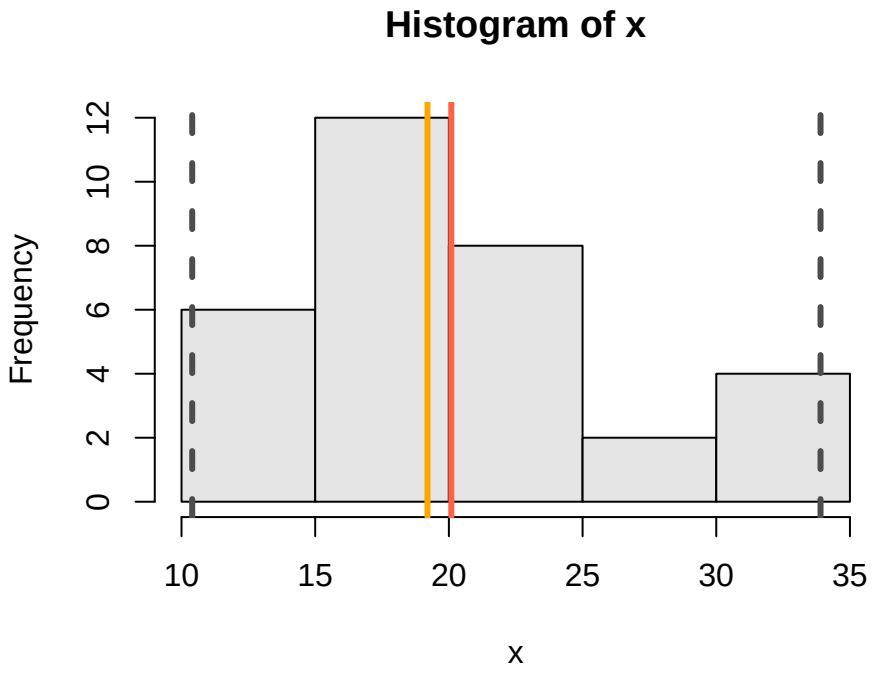
Write a function `histogram()` that plots a histogram with added vertical lines for the following summary statistics: minimum value, median, mean, and maximum value. The main idea is to wrap the high-level function `hist()` and then plot the lines with a low-level plotting function.

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

For instance:

```
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()`

For example:

```
to_celsius(34)
```

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

```
to_kelvin(34)
```

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

```
to_reaumur(34)
```

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

```
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`, like this:

```
convert(40, to = "celsius")
```

By default, `to = "celsius"`, but it can take values such as `"kelvin"`, `"reaumur"`, or `"rankine"`

For instance:

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

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

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

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

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

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

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

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

Permutations

The possible number of combinations of k objects from a set of n objects is given by the formula:

$$nCk = \frac{n!}{k!(n-k)!}$$

where $n!$ is the factorial of a number n, i.e. $n! = n(n-1)(n-2)...(2)(1)$. For instance, the number of combinations of 2 objects from a set of 4 objects is:

$$\binom{4}{2} = \frac{4!}{2!(4-2)!} = 6$$

R provides the functions `factorial()` to compute the factorial of a number:

```r
factorial(4)
```

```
## [1] 24
```

R also provides the function `choose()` that computes the number of combinations:

```r
# combinations of 2 objects from a set of 4
choose(4, 2)
```

```
## [1] 6
```

R, however, does not have a function to compute permutations:

$$_nP_k = \frac{n!}{(n-k)!}$$

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

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

Average function with for loop

R provides the function `mean()` to calculate the arithmetic mean (i.e. average) of a numeric object. Create a function `average()` using a ***for loop*** to compute the mean. `average()` takes a numeric vector and returns the average.

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

For instance:

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

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

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

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

Note: Remember that all arithmetic and math functions, as well as logical comparisons, are vectorized. This example with the `average()` function is just for practicing using control-flow structures.

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***:

For instance:

```r
geomean(1)
```

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

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

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

Note: Again, keep in mind that this example is for practicing purposes. In R, when performing a computation that involves iterating through the elements of a vector, a for loop might not be the best option. Instead of using a for loop, you can use the function `prod()` which is already vectorized. Here's how:

```
gmean <- function(x) {
  prod(x)^(1 / length(x))
}
```

`gmean()` will be more efficient than any function we create using for loops:

```
gmean(1)
```

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

```
gmean(1:5)
```

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

```
gmean(seq(-1, 1, length.out = 20))
```

```
## [1] 0.4007307
```

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.

Here's an example of how the output should look like:

```
# 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.

For example:

```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"`

For instance:

```
# 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
```

```
# 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:

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