Stat 133 HW3: Flow Control Structures and Functions with R

Instructor: 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 code chunk.

Arithmetic average with loops

R provides the function `mean()` that can be applied to calculate the arithmetic mean (i.e. average) of a numeric vector. Here's the formula of the average for a vector $x = (x_1, x_2, \dots, x_n)$

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

Use the three types of loops—`for, repeat, while`—to get the average of the numeric vector `x <- 1:10`

```
# your for loop



# your repeat loop



# your while loop



#
```

Area of a circle

For a given circle of radius r, the area A is:

$$A = \pi r^2$$

Write a function `circle_area()` that calculates the area of a circle. This function must take one argument `radius`. Give `radius` a default value of 1.

For example:

```
# default (radius 1)
circle_area()
```

```
## [1] 3.141593
```

```
# radius 3
circle_area(radius = 3)
```

```
## [1] 28.27433
```

Area of a cylinder

For a given cylinder of radius r and height h the area A is:

$$A = 2\pi rh + 2\pi r^2$$

Notice that the formula of the area of a cylinder includes the area of a circle: πr^2. Write a function `cyl_area()`, that calls `circle_area()`, to compute the area of a cylinder. This function must take two arguments: `radius` and `height`. Give both arguments a default value of 1.

For instance:

```
# default (radius 1, height 1)
cyl_area()
```

```
## [1] 12.56637
```

```
# radius 2, height 3
cyl_area(radius = 2, height = 3)
```

```
## [1] 62.83185
```

Volume of a cylinder

For a given cylinder of radius r and height h the volume V is:

$$V = \pi r^2 h$$

Write a function `cyl_volume()`, that calls `circle_area()`, to compute the volume of a cylinder. This function must take two arguments: `radius` and `height`. Give both arguments a default value of 1.

For example:

```
# default (radius 1, height 1)
cyl_volume()
```

```
## [1] 3.141593
```

```r
cyl_volume(radius = 3, height = 10)
```

```
## [1] 282.7433
```

```r
cyl_volume(height = 10, radius = 3)
```

```
## [1] 282.7433
```

Even number

Write a function `is_even()` that determines whether a number is even (i.e. multiple of 2). If the inpute number is even, the output should be `TRUE`. If the input number is odd, the output should be `FALSE`. If the input is not a number, the output should be `NA`

For example:

```r
# even number
is_even(10)
```

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

```r
# odd number
is_even(33)
```

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

```r
# not a number
is_even('a')
```

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

Odd number

Use your function `is_even()` to write a function `is_odd()` that determines if a number is odd (i.e. not a multiple of 2). If a number is odd, the output should be `TRUE`; if a number is even the output should be `FALSE`; if the input is not a number the output should be `NA`

For example:

```r
# odd number
is_odd(1)
```

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

```r
# even number
is_odd(4)
```

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

```r
# not a number
is_odd('a')
```

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

Quadratic Formula

One way to find the real roots of a 2nd degree polynomial, $ax^2 + bx + c$, is by using the famous quadratic equation:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Write a function `quadratic()` that solves the quadratic equation. It should take three arguments: `a, b, c` corresponding to the coefficients of the polynomial. Give default values `a=1, b=1, c=0`. The output must be a list with two elements `$sol1` and `$sol2` containing the two roots:

For example:

```r
# default
quadratic(a = 1, b = 1, c = 0)
```

```
## $sol1
## [1] 0
##
## $sol2
## [1] -1
```

```r
# x^2 + 2b - 2
quadratic(c = -2, b = 2, a = 1)
```

```
## $sol1
## [1] 0.7320508
##
## $sol2
## [1] -2.732051
```

Not all polynomials of 2nd degree have real roots—some may have complex roots. For example, if you apply `quadratic()` to solve $x^2 + 2x + 2$, you should get the following result:

```r
quadratic(a = 1, b = 2, c = 2)
```

```
## Warning in sqrt(b^2 - 4 * a * c): NaNs produced
```

```
## $sol1
## [1] NaN
##
## $sol2
## [1] NaN
```

In order to limit the results to just real roots, it would be nice to have a function that indicates that there are no real roots. Use your `quadratic()` function to write a function `find_roots()` that either: a) finds the real roots, or b) returns a message `"No real roots"` if there are no real roots

For example:

```
find_roots(a = 1, b = -1, c = -2)
```

```
## $sol1
## [1] 2
##
## $sol2
## [1] -1
```

```
find_roots(a = 1, b = 2, c = 2)
```

```
## [1] "No real roots"
```

Converting Miles to other units

The table below shows the different formulas for converting miles (mi) into other scales:

Units	Formula
Inches	mi x 63360
Feet	mi x 5280
Yards	mi x 1760
Meters	mi / 0.00062137
Kms	mi / 0.62137

Write the following five functions for each type of conversion. Each function must take one argument `x` with default value: `x = 1`.

- `miles2inches()`
- `miles2feet()`
- `miles2yards()`
- `miles2meters()`
- `miles2kms()`

For example:

```
miles2inches(2)
```

```
## [1] 126720
```

```
miles2feet(2)
```

```
## [1] 10560
```

```r
miles2yards(2)
```

```
## [1] 3520
```

```r
miles2meters(2)
```

```
## [1] 3218.694
```

```r
miles2kms(2)
```

```
## [1] 3.218694
```

Using switch()

Create a function `convert()` that converts miles into the specified units. Use `switch()` and the previously defined functions—`miles2inches()`, miles2feet(), ..., `miles2kms`—to define `convert()`. Use two arguments: `x` and `to`, like this:

```r
convert(40, to = "in")
```

By default, `to = "km"`, but it can take values such as `"in"`, `"ft"`, `"yd"`, or `"m"`.

For instance:

```r
convert(3, "in")
convert(3, "ft")
convert(3, "yd")
convert(3, "m")
convert(3, "km")
```

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.

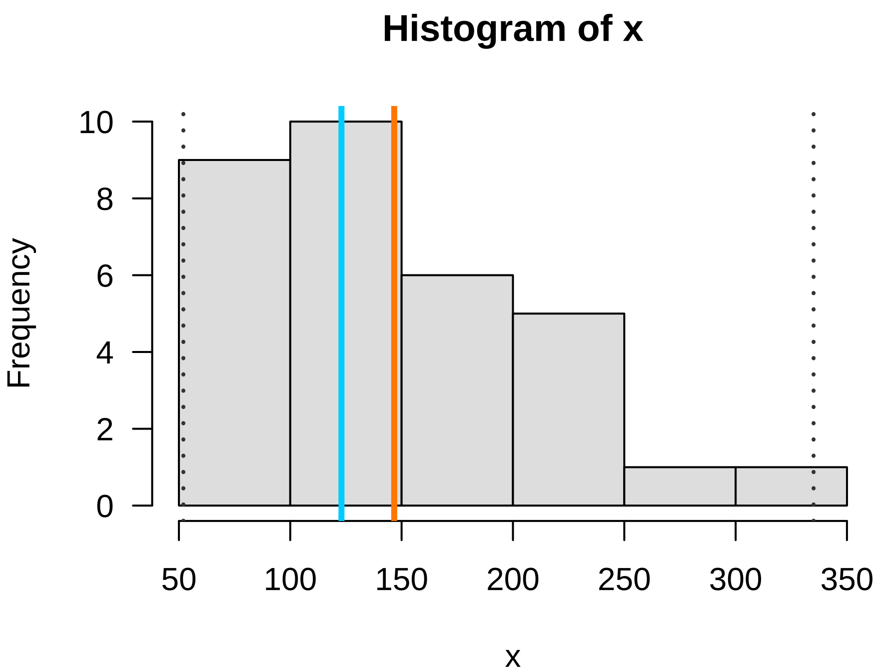
Define your function with the following requirements:

- bars of histogram colored in `"#DDDDDD"`
- line of minimum value in color `"#333333"`, and dotted type
- line of maximum value in color `"#333333"`, and dotted type
- line of median value in color `"#00CCFF"`
- line of mean value in color `"#FF7502"`
- max and min lines with a width of 2
- median and mean lines with a width of 3

- axis tick labels in horizontal orientation

For instance:

```
histogram(mtcars$hp)
```



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) `frequency`: absolute frequency
3) `proportion`: relative frequency (round to four decimal digits)
4) `cumfreq`: cumulative absolute frequency
5) `cumprop`: cumulative relative frequency (round to four decimal digits)

`freq_table()` must have two arguments: `x` and `decreasing`. The argument `x` is the provided factor. The argument `decreasing` indicates whether to return the results of the table in decreasing or increasing order. Give `decreasing` a default value of `NULL`.

By default `decreasing = NULL` means that the table is returned as is. `decreasing = TRUE` indicates that the results are displayed by frequency in decreasing order; `decreasing = FALSE` indicates that the results are displayed by frequency in increasing order. Make sure that the input is a factor (otherwise the function should convert the input into factor). Likewise, the output should be in `data.frame` format.

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

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

# frequency table (default)
freq_table(sizes)
```

```
##   category frequency proportion cumfreq 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
# frequencies in decreasing order
freq_table(sizes, decreasing = TRUE)
```

```
##   category frequency proportion cumfreq cumprop
## 1   medium        40     0.4444      40  0.4444
## 2    small        27     0.3000      67  0.7444
## 3    large        23     0.2556      90  1.0000
```

```r
# frequencies in increasing order
freq_table(sizes, decreasing = FALSE)
```

```
##   category frequency proportion cumfreq cumprop
## 1    large        23     0.2556      23  0.2556
## 2    small        27     0.3000      50  0.5556
## 3   medium        40     0.4444      90  1.0000
```