Workout 2 - Testing Functions and Strings

Stat 133, Fall 2018, Prof. Sanchez

Due date: Nov-06 (before midnight)

In this assignment, you will create a set of different functions, applying the concetps covered so far: creating functions, documenting functions with Roxygen comments, using if-conditionals, loop structures, and manipulation of strings. In addition, you will also have to implement tests for your functions using the R package `"testthat"`.

All your functions should include Roxygen comments: e.g.

- `@title`
- `@description`
- `@param`
- `@return`

1) File Structure

After completing this assignment, the file structure should look like this:

```
workout02/
   README.md
   code/
      functions/
         minkowski.R
         hex-color.R
         reverse-chars.R
         count-vowels.R
      tests/
         test-minkowski.R
         test-hex-color.R
         test-reverse-chars.R
         test-count-vowels.R
   output/
      run-tests.R
      test-output.txt
```

- Use Git to *add* and *commit* the changes as you progress with your HW.
- And don't forget to *push* your commits to your github classroom repository.
- **Submit the link of your github classroom repository to bCourses. Do NOT submit any files (we will actually turn off the uploading files option).**

2) Create a `README.md` File

Create a `README.md` file and include a description of what the HW is about. Recall that the content of this file should let you (or any other user) get an idea of what this assignment is about.

3.1) Minkowski Distance

For a point $x = (x_1, x_2, \ldots, x_n)$ and a point $y = (y_1, y_2, \ldots, y_n)$, the Minkowski distance of order p (p-norm distance) is defined as:

$$\text{1-norm distance} = \sum_{i=1}^{n} |x_i - y_i|$$

$$\text{2-norm distance} = \left(\sum_{i=1}^{n} |x_i - y_i|^2 \right)^{1/2}$$

$$\text{p-norm distance} = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

$$\text{infinity norm distance} = max(|x_1 - y_1|, |x_2 - y_2|, \ldots, |x_n - y_n|)$$

where p need not be an integer, but it cannot be less than 1, because otherwise the triangle inequality does not hold. In case you're curious, you can find more information about this type of distance in the following wikipedia entry:

https://en.wikipedia.org/wiki/Distance

Function `minkowski()`

In the subdirectory `functions/`, create an R script file `minkowski.R` in order to write a function `minkowski()`—including roxygen comments—for the Minkowski distances.

To compute the summation terms you cannot use vectorized code. In other words, you have to use some sort of loop structure (e.g. `for()` loop).

- The function should take three arguments:
  - `x` = numeric vector for one point
  - `y` = numeric vector for the other point
  - `p` = either a numeric value greater than 1, or a character string `max` (default 1)

- It should return the computed distance. Do NOT use `print()` for the returned output. Use `return()` instead.
- Check that `x` and `y` have the same length, otherwise raise an error: `"x and y have different lengths"`.
- Give `p` a default value of 1.
- If `p` is numeric, check that `p` is greater than 1, otherwise raise an error: `"p cannot be less than 1"`
- If `p` is character, check that `p` equals `"max"`, otherwise raise an error: `"invalid character value for p"`

Tests for `minkowski()`

Inside the `tests/` subdirectory, create a file `test-minkowski.R` with the following content:

```
context("Tests for minkowski()")


test_that("minkowski works as expected", {
  point1 <- c(0, 0)
  point2 <- c(1, 1)
  point3 <- sqrt(c(2, 2))
  point4 <- c(0, 1)
  point5 <- c(1, 1, 1)

  expect_equal(minkowski(point1, point2, p = 1), 2)
  expect_equal(minkowski(point1, point3, p = 2), 2)
  expect_equal(minkowski(point1, point2, p = 'max'), 1)

  expect_length(minkowski(point1, point2, p = 1), 1)
  expect_type(minkowski(point1, point2, p = 1), 'double')

  expect_error(minkowski(point4, point5, p = 1))
  expect_error(minkowski(point1, point2, p = 0.5))
  expect_error(minkowski(point1, point2, p = 'min'))
})
```

3.2) Colors in Hexadecimal Notation

Recall that a hexadecimal color starts with a hash `#` symbol followed by six hexadecimal digits: 0 to 9, and the first six letters A, B, C, D, E, F. For eaxmple, the color red in hex notation is: `#FF0000`.

Functions `is_hex()` and `is_hex_alpha()`

In the subdirectory `functions/`, create an R script file `hex-color.R` in order to write two functions: `is_hex()` and `is_hex_alpha()`, to check whether an input string is a valid color in hexadecimal notation.

`is_hex()` checks whether an input string is a valid hex color without an *alpha* transparency value. In contrast, `is_hex_alpha()` checks whether an input string is a valid hex color with an *alpha* transparency value (e.g. `#6083E1A3`). If the input is a valid hex color, then the output should be `TRUE`, otherwise `FALSE`.

If the input is not a string, raise an error: `"invalid input; a string was expected"`.

Since R accepts hex-colors with lower case letters (a, b, c, d, e, f) your function should work with both upper and lower case letters.

Tests for `is_hex()` and `is_hex_alpha()`

Inside the `tests/` subdirectory, create a file `test-hex-color.R` with the following content:

```
context("Test for is-hex-color")


test_that("is_hex() works as expected", {

  expect_true(is_hex("#FF00A7"))
  expect_true(is_hex("#ff0000"))
  expect_true(is_hex("#123456"))
  expect_true(is_hex("#12Fb56"))

  expect_false(is_hex("FF0000"))
  expect_false(is_hex("#1234GF"))
  expect_false(is_hex("#1234567"))
  expect_false(is_hex("blue"))

  expect_error(is_hex(FF00A7))
  expect_error(is_hex(TRUE))
})



test_that("is_hex_alpha() works as expected", {

  expect_true(is_hex_alpha("#FF000078"))
  expect_true(is_hex_alpha("#ffda0078"))
  expect_false(is_hex_alpha("#FF0000"))
```

```
  expect_false(is_hex_alpha("#ffda00"))

  expect_error(is_hex_alpha(FF00A7))
  expect_error(is_hex_alpha(TRUE))
})
```

3.3) Reversing Characters

In the subdirectory `functions/`, create an R script file `reverse-chars.R` in order to write a function `reverse_chars()` that reverses a string by characters.

For instance, if you have a string:

`"Was it a car or a cat I saw?"`

then `reverse_chars()` should return:

`"?was I tac a ro rac a ti saW"`

Tests for `reverse_chars()`

Inside the `tests/` subdirectory, create a file `test-reverse-chars.R` with the following content:

```
context("Test for reverse_chars")


test_that("reverse_chars() works as expected", {
  pets <- "step on no pets"
  ep <- "expecto patronum"
  pe <- "munortap otcepxe"
  car_cat <- "Was it a car or a cat I saw?"
  tac_rac <- "?was I tac a ro rac a ti saW"

  expect_equal(reverse_chars(pets), pets)
  expect_equal(reverse_chars(ep), pe)
  expect_equal(reverse_chars(car_cat), tac_rac)
  expect_length(reverse_chars(car_cat), 1)
  expect_type(reverse_chars(car_cat), "character")
  expect_equal(nchar(reverse_chars(car_cat)), nchar(reverse_chars(tac_rac)))
})
```

3.4) Counting Vowels

In the subdirectory `functions/`, create an R script file `count-vowels.R` in order to write a function `count_vowels()` that computes the number of vowels of a character string.

If the input is not a string, raise an error: `"invalid input; a string was expected"`.

Make sure that `count_vowels()` counts vowels in both lower and upper case letters.

For instance, if you have a string:

`"The quick brown fox jumps over the lazy dog"`

then `count_vowels()` should return a numeric vector (with names for vowels):

```
a e i o u
1 3 1 4 2
```

Likewise, if you have the following string:

`"THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG"`

then `count_vowels()` should return the same numeric vector (with names for vowels) as above:

```
a e i o u
1 3 1 4 2
```

Tests for `count_vowels()`

Inside the `tests/` subdirectory, create a file `test-count-vowels.R` with the following content:

```r
context("Tests for count vowels")


test_that("count_vowels works as expected", {
  fox <- "The quick brown fox jumps over the lazy dog"
  FOX <- "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG"
  do <- "do or do not"
  vowels <- c('a', 'e', 'i', 'o', 'u')
  counts <- c(1, 3, 1, 4, 2)
  names(counts) <- vowels

  expect_equal(count_vowels(fox), counts)
  expect_equal(count_vowels(FOX), counts)
  expect_length(count_vowels(fox), 5)
  expect_named(count_vowels(fox), vowels)
  expect_type(count_vowels(fox), 'double')
  expect_length(count_vowels(do), 5)
```

```
  expect_error(count_vowels(123))
  expect_error(count_vowels(TRUE))
})
```

4) Running Tests

In the subdirectory `output/` create an R script file `run-tests.R` with the content displayed below. This is the *master* script that you will execute to run all the tests. Make sure to include a meaningful header. By the way, you should run this script assuming that your working directory is the directory `output/`.

```
# include a header!

library('testthat')

functions <- dir('../code/functions')
lapply(paste0('../code/functions/', functions), source)

sink(file = 'test-output.txt')
test_dir('../code/tests')
sink()
```