Warm-Up 05 - Functions

Stat 133, Fall 2018, Prof. Sanchez

Due date: Oct-16 (before midnight)

The purpose of this assignment is to begin working on the programming concepts that are covered in week 7:

- writing simple functions
- documenting functions with Roxygen comments
- using conditionals

General Instructions

- Write your narrative and code in an `Rmd` (R markdown) file.
- Name this file as `warmup05-first-last.Rmd`, where `first` and `last` are your first and last names (e.g. `warmup05-gaston-sanchez.Rmd`).
- Include a code chunk at the top of your file like the one in the following screen capture:

````
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE, error = TRUE)
```
````

- Since you willl be writing a couple of functions with `stop()` statements, it is essential that you set up `error = TRUE`, otherwise `"knitr"` will stop knitting your `Rmd` file if it encounters an error.
- All your functions should include Roxygen comments: e.g.
  - `@title`
  - `@description`
  - `@param`
  - `@return`
- Submit your Rmd and html files to bCourses.

---

1) Gaussian Function

The Gaussian (Normal) function, given in the equation below, is one of the most widely used functions in science and statistics:

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

The parameters σ and μ are real numbers, where σ must be greater than zero. For more information, see the wikipedia entry:

https://en.wikipedia.org/wiki/Normal_distribution#/media/File:Normal_Distribution_PDF.svg

Make a function `gaussian()` that takes three arguments: `x`, `m`, and `s`. Evaluate the function with $m = 0$, $s = 2$, and $x = 1$.

Test your `gaussian()` function and compare it with the R function `dnorm()`

```r
# compare with dnorm()
dnorm(x = 1, mean = 0, sd = 2)
```

```
## [1] 0.1760327
```

Once you have your `gaussian()` function try it with a vector `seq(-4.5, 4.5, by = 0.1)`, and pass the values to `plot()` to get a normal curve. Here's some code with values obtained from `dnorm()`

```r
# gaussian curve
x_values <- seq(from = -4.5, to = 4.5, by = 0.1)
y_values <- dnorm(x_values, mean = 0, sd = 2)
plot(x_values, y_values, las = 1, type = "l", lwd = 2)
```

In addition to the above plot, you should also try to replicate—as much as possible—the following graph (original version displayed in the wikipedia entry of the normal distribution). You can use any plotting approach, but you must specify colors in hexadecimal notation.

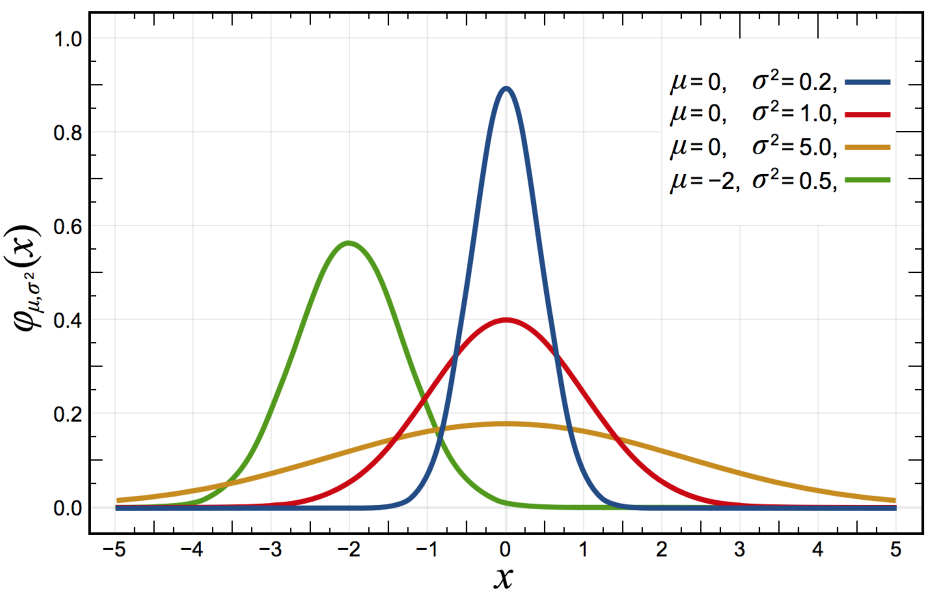

Figure 1: Normal probability density functions

Here are some resources about how to add mathematical symbols in R plots:

- https://andyphilips.github.io/blog/2017/08/16/mathematical-symbols-in-r-plots.html
- https://trinkerrstuff.wordpress.com/2018/03/15/2246/

2) Descriptive Statistics

Write a function `descriptive()` that takes a numeric vector as input, and returns a **named vector** with the following descriptive statistics:

- `min`: minimum
- `q1`: first quartile (Q2)
- `median`: median
- `mean`: mean
- `q3`: third quartile (Q3)
- `max`: maximum
- `range`: range or span (max - min)
- `iqr`: interquartile range (IQR)
- `sd`: standard deviation

The function `descriptive()` should `stop()`—with a descriptive error message—when the input vector is not numeric. Also, it should have a parameter `na.rm` that allows the user to indicate if missing values should be removed before computation. By default, `na.rm = FALSE`.

Test the function with the following code:

```
# input vectors
set.seed(100)
x <- rnorm(100)
y <- x
y[sample(1:100, size = 10)] <- NA

# try your function
descriptive(x)
descriptive(y)
descriptive(y, na.rm = TRUE)
descriptive(letters)
```

3) Two Given Points

Let p_1 and p_2 be two points with two coordinates: $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$.

The distance d between two points can be calculated with the formula:

$$d = \sqrt{(x_2 - x_2)^2 + (y_2 - y_1)^2}$$

The midpoint of the line segment between p_1 and p_2 can be found as:

$$p = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right)$$

The intercept a and the slope b of the line $y = a + bx$ connecting two points p_1 and p_2 can be found as:

$$b = \frac{y_2 - y_1}{x_2 - x_1}, \quad a = y_1 - bx_1$$

Distance

Write a function `find_distance()` that returns the distance between two given points. You should be able to call the function like this:

```
# coordinates for point-1 and point-2
p1 <- c(0, 0)
p2 <- c(1, 1)

find_distance(p1, p2)
```

Midpoint

Write a function `find_midpoint()` that returns the midpoint between two given points. You should be able to call the function like this:

```
p1 <- c(0, 0)
p2 <- c(1, 1)

find_midpoint(p1, p2)
```

Slope

Write a function `find_slope()` that returns the slope of the line connecting two given points. You should be able to call the function like this:

```
p1 <- c(0, 0)
p2 <- c(1, 1)

find_slope(p1, p2)
```

Intercept

Write a function `find_intercept()` that returns the intercept of the line connecting two given points. This function must internally use `find_slope()`

```r
p1 <- c(0, 0)
p2 <- c(1, 1)

find_intercept(p1, p2)
```

Line

Write a function `find_line()`. This function must use `find_slope()` and `find_intercept()`. The output should be a list with two named elements: `"intercept"` and `"slope"`, Here is how you should be able to use `find_line()`:

```r
p1 <- c(0, 0)
p2 <- c(1, 1)

eq <- find_line(p1, p2)
eq$intercept
eq$slope
```

Information about two given points

Once you have the functions `find_distance()`, `find_midpoint()`, and `find_line()`, write an overall function called `info_points()` that returns a list with the distance, the midpoint, and the line's slope and intercept terms. Here is how you should be able to use `info_points()`:

```r
p1 <- c(-2, 4)
p2 <- c(1, 2)

results <- info_points(p1, p2)
results$distance
results$midpoint
results$intercept
results$slope
```

Use the following code to create a plot that displays the given points, the line, and the midpoint. Note that the title of the plot shows the line equation. For instance, if the points are $p_1 = (-2, 4)$ and $p_2 = (1, 2)$, the plot may look like this (you should choose different points!):

```r
# change these points and pass them to info_point()
p1 <- c(-2, 4)
p2 <- c(1, 2)

plot.new()
# depending on your chosen points you may have to set different limits
plot.window(xlim = c(-3, 3), ylim = c(0, 5))
```

```r
axis(side = 1)
axis(side = 2, las = 1)
points(p1[1], p1[2], cex = 1.5, col = "#FF8834", pch = 19)
points(p2[1], p2[2], cex = 1.5, col = "#FF8834", pch = 19)
# midpoint (here you should use the midpoint outputs of your function)
points(-1/2, 3, cex = 1.5, pch = "x", col = "#E16868")
# slope and intercept (here you should use the outputs of your function)
abline(a = 8/3, b = -2/3, col = "#FF883477", lwd = 3)
title(main = expression(paste(y, ' = ', (-2/3) * x, ' + ', (8/3))))
```