Homework 2

Student ID: 39-246182

Student name: GONG Kuiyuan

Preferred name: Eddie

Answers:

Task 1: In R, we should use `?` to check the documentation for the function that we don't understand. In the case of `lm`, we can input the following code in console to get the documentation we need.

```
?lm
```

Task 2: It is important to know that `lm` function is used for linear models which could include multiple variate. Basically, `formula` refers to the composition of regression function. `data` refers to the data used for such regression. `subset` allows us to restrict the regression to certain groups of samples. There are other arguments like `weights` in `lm`.

```
lm (formula, data, subset, weights...)
```

```
lm (y ~ x1 + x2, data = data_1, subset = y > 10000)
```

Besides, in order to create data used for such regression, we may use `data.frame()` to do this.

```r
data_1 <- data.frame(
  x1 <- 1:100,
  x2 <- 2:101,
  y <- 3:102
)

result_1 <- lm (y ~ x1 + x2, data = data_1, subset = y > 50)

print(result_1)
```

```
Call:
lm(formula = y ~ x1 + x2, data = data_1, subset = y > 50)

Coefficients:
(Intercept)           x1           x2
          2            1           NA
```

Task 3: Using the above example. `formula` needs to be supplied by the user. `data`, `subset` and `weights` are in default if there is no specification. For example, if we didn't specify `data = data_1` and there are values of `x1`, `x2` and `y` in the global environment initially. The `lm` function would use values from the global environment to run the regression.

Task 4: *Rosenbrock's banana function:*

```r
a <- 1
b <- 100

Rosenbrock <- function(x, y){
  result_2 <- (a - x)^2 + b*(y - x^2)^2
  return(result_2)
}

Rosenbrock (99,99) # Could take any values of x and y.
```

```
[1] 9412890004
```

```r
result_2 <- Rosenbrock (99,99)
print (result_2)
```

```
[1] 9412890004
```

Task 5: When we call the arguments by name, their positions in the function do not matter anymore. The following two functions return the same result.

```
Rosenbrock(2,3) # Return a result of 101.
```

```
[1] 101
```

```
Rosenbrock(y = 3, x = 2) # After changing the position, the result doesn't change since we ca
```

```
[1] 101
```

Task 6: In this part, I tried not passing the values of `a` and `b` at `function()`, even though there are values for `a` and `b` in the global environment. The function doesn't use it until I pass some values for these two argument within `function()`. Going back to the task itself, `Rosenbrock2 (1, 1)` will return a result using the default values of `a` and `b` unless passing particular values like `Rosenbrock2 (1, 1, 2, 2)`.

```
Rosenbrock2 <- function(x, y, a = 1, b = 100){
  result_3 <- (a - x)^2 + b*(y - x^2)^2
  return(result_3)
}

Rosenbrock2 (1, 1) # Without passing the values.
```

```
[1] 0
```

```
Rosenbrock2 (1, 1, 2, 2) # Passing the values of a and b.
```

```
[1] 1
```

```
result_3 <- (Rosenbrock2 (1, 1, 2, 2))
print(result_3)
```

```
[1] 1
```

Task 7: Rosenbrock's banana function written in LaTex:

$$f(x, y) = (a - x)^2 + b \cdot (y - x^2)^2$$

Task 8: *R's lazy evaluation:*

```
Lazy <- function(x,y){
  result_4 <- x^2
  return (result_4)
}

Lazy(2) # The behavior of the function is not affected even if the value of y is missing.
```

```
[1] 4
```

```
result_4 <- Lazy(2)
print(result_4)
```

```
[1] 4
```

Task 9: The function `class()` is used to check which class is the target belonged to. However, I found that `class()` is different from `typeof()`.

```
?class

a <- 1

class (a) # returning "numeric".
```

```
[1] "numeric"
```

```
typeof (a) # returning "double", which is different from the above.
```

```
[1] "double"
```

```
x1 <- 1:100

class (x1)
```

```
[1] "integer"
```

```
typeof (x1) # In this case, both return "integer".
```

```
[1] "integer"
```

Task 10:

```
class (Lazy)
```

```
[1] "function"
```

```
class (Rosenbrock) # Returning "function"
```

```
[1] "function"
```

```
z <- 'Econ'
class (z)
```

```
[1] "character"
```