Hw_2 Data 412-612

Ethan Pastel

2023-09-21

Question 1) Write a buzzfizz() Function. Create a new function which takes a single number as input. Use logicals and conditionals as well as the operator %% as required.

a. Design your function. Write out in words in your .Rmd file, the logic of the steps your function will need to accomplish to convert the input number into the specified output.

Step One: First I will create a function named buzzfizz and assign it to (function(number())), which allows the user to input a test variable at their discretion.

Step Two: Next I will create if and else if statements to check whether the the number is divisible by 5 and if it is I will have R return 'fizz'.

Step Three: I will then use if and else if statements to check whether the the number is divisible by 3 and if it is I will have R return 'buzz'.

Step Four: I will contuine to use if and else if statements to check whether the the number is divisible by 3 and 5. If it is I will have R return 'buzzfizz'.

Step Five: Finally, at the end of the if else statements I will return the inputted number if it isn't divisible by one of the following: 3, 5, and 3 & 5.

b. Write R code (Not a function) in a code chunk to implement and test the logic of each of your steps individually on a variable x. Test it with x having values of of 3, 5, 15, and 16.

```r
buzzfizz <- function(number) {
  if (number %% 5 == 0)
    return("buzz")

    else if (number %% 3 == 0)
    return("fizz")

    else if (number %% 3 == 0 && number %% 5 == 0)
    return("buzzfizz")

    else
    return(number)
}

print(buzzfizz(3))
```

```
## [1] "fizz"
```

```r
print(buzzfizz(5))
```

```
## [1] "buzz"
```

```r
print(buzzfizz(15))
```

```
## [1] "buzz"
```

```r
print(buzzfizz(16))
```

```
## [1] 16
```

c. Once the individual steps are working, create a new code chunk and copy the code using variable x to the new code chunk and turn it into function with the input argument of x. Make sure x has a default argument

```r
buzzfizz <- function(x) {
  if (x %% 5 == 0)
    return("buzz")

    else if (x %% 3 == 0)
    return("fizz")

    else if (x %% 3 == 0 && x %% 5 == 0)
    return("buzzfizz")

    else
    return(x)
}
```

d. Show your output for the following inputs: 3, 5, 15, 2.

```r
print(buzzfizz(3))
```

```
## [1] "fizz"
```

```r
print(buzzfizz(5))
```

```
## [1] "buzz"
```

```
print(buzzfizz(15))
```

```
## [1] "buzz"
```

```
print(buzzfizz(2))
```

```
## [1] 2
```

Copy and paste your last function into a new code chunk. Update your function to include error checking. Include checks to ensure the input is both numeric and a single value, not a vector. Test it on inputs cat, and c(1,5). In the code chunk where you run the function, set your code chunk parameter for error to be TRUE.

```
buzzfizz <- function(x) {
  if (!is.numeric(x))
    stop("Input must be numeric.")

  if (length(x) != 1)
    stop("Input cannot be a vector.")


  if (x %% 5 == 0)
    return("buzz")

    else if (x %% 3 == 0)
    return("fizz")

    else if (x %% 3 == 0 && x %% 5 == 0)
    return("buzzfizz")

    else
    return(x)
}

buzzfizz(c(1,5))
```

```
## Error in buzzfizz(c(1, 5)): Input cannot be a vector.
```

```
buzzfizz(15)
```

```
## [1] "buzz"
```

```
buzzfizz("cat")
```

```
## Error in buzzfizz("cat"): Input must be numeric.
```

f. Copy and paste your function with error checking into a new code chunk. Complete your function by inserting and completing Roxygen2 comments in the code chunk, above the function, to document the function. Include the following elements: title, description, usage, arguments (the params), and return value. Include three examples.

```
library(roxygen2)
```

```
## Warning: package 'roxygen2' was built under R version 4.3.2
```

```
#' BuzzFizz Function
#'
#' This function takes a numeric input and returns "buzz" if the input is divisible by 5,
#' "fizz" if it is divisible by 3, and then "buzzfizz" if it is divisible by both 3 and 5,
#' and the input itself otherwise.
#'
#' @param x Numeric input.
#'
#' @return Returns "buzz", "fizz", "buzzfizz", or the input itself.
#'
#' @examples
#' buzzfizz(2) # Returns 2
#' buzzfizz(15) # Returns "buzzfizz"
#' buzzfizz(10) # Returns "buzz"
#'
#' @export


buzzfizz <- function(x) {
  if (!is.numeric(x))
    stop("Input must be numeric.")

  if (length(x) != 1)
    stop("Input cannot be a vector.")


  if (x %% 5 == 0)
    return("buzz")

    else if (x %% 3 == 0)
    return("fizz")

    else if (x %% 3 == 0 && x %% 5 == 0)
    return("buzzfizz")

    else
    return(x)
}

buzzfizz(c(1,5))
```

```
## Error in buzzfizz(c(1, 5)): Input cannot be a vector.
```

```
buzzfizz(15)
```

```
## [1] "buzz"
```

```
buzzfizz("cat")
```

```
## Error in buzzfizz("cat"): Input must be numeric.
```

2. Using a Forward Pipe

1. Create a code chunk and set the random number seed to 1 using set.seed(1). In the same code chunk, use a forward pipe, %>% or |>, to do the following steps without intermediate variables: Use sample() to sample from the vector 1:10 1000 times with replacement. Calculate the mean of the resulting sampled vector, then Exponentiate that mean.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
set.seed(1)

end_mean <- 1:10 %>%
  sample(1000, replace = TRUE) %>%
  mean() %>%
  exp()

print(end_mean)
```

```
## [1] 287.4359
```

3. Calculate a Proportion

Set the seed to 1 and select a random sample of 100 normally distributed values with mean 10 and variance of 3. Calculate the proportion greater than 12.

```
set.seed(1)

random_samp <- rnorm(100, mean = 10, sd = sqrt(3))
```

```
proportion_greater_than_12 <- mean(random_samp > 12)

print(proportion_greater_than_12)
```

```
## [1] 0.12
```

Logical Comparisons and Subsetting

Create the values:

– x <- c(TRUE, FALSE, TRUE, TRUE)

– y <- c(FALSE, FALSE, TRUE, FALSE)

– z <- NA

```
x <- c(TRUE, FALSE, TRUE, TRUE)
y <- c(FALSE, FALSE, TRUE, FALSE)
z <- NA
```

What are the results of the following:

– x & y

– x & z

– !(x | y)

– x | y

– y | z

– x[y]

– y[x]

– x[x|y

```
x & y
```

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

```
x & z
```

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

```r
!(x | y)
```

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

```r
x | y
```

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

```r
y | z
```

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

```r
x[y]
```

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

```r
y[x]
```

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

```r
x[x|y]
```

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