Homework4

Sherine George

30 September, 2023

QQ1.Install (if not available already) and load the httr and jsonlite libraries. Copy, paste, and execute the code included in the ip_address2country_code.r file, which you will need to use in order to solve this exercise. Read the logins.csv dataset, which contains data about the times at which a suspicious user logged in to a given website. The data is already ordered by utc_time, the column indicating the time of each login. Add to the tibble the following three additional columns: – last_seen, representing the number of full days that occurred between successive logins; hint: ? lag, and use as.integer to cast the final result to an integer value. – is_weekend, representing a logical column (i.e., a column with values TRUE or FALSE) that indicates whether the login occurred on a weekend (i.e., on Saturday or Sunday); hint: ? weekdays. – country_code, representing the country from which the connection originated; hint: use the ip_address2country_code function above to determine the country associated with a given IP address.

code included in the ip_address2country_code.r file :

```r
ip_address2country_code <- function(ip_addresses) {
  ip_api_url <- "http://ip-api.com/batch/"
  df <- data.frame(query = ip_addresses)
  request_body <- toJSON(df)
  response <- POST(url = ip_api_url, body = request_body)
  response_content <- content(response)
  country_codes <-
    sapply(response_content, function(x)
      ifelse(is.null(x$countryCode), NA_character_, x$countryCode))
  return(country_codes)
}
```

load data set

```r
logins <- read_csv("C:\\Users\\Sherine\\Downloads\\logins.csv")
```

```
## Rows: 10 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (1): ip_address
## dttm (1): utc_time
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
logins <- logins %>%
  mutate(
    last_seen = as.integer(difftime(utc_time, lag(utc_time), units = "days")),
    is_weekend = ifelse(weekdays(utc_time) %in% c("Saturday", "Sunday"),"TRUE","FALSE"),
    country_code = sapply(ip_address, ip_address2country_code)
  )
logins
```

```
## # A tibble: 10 x 5
##    ip_address      utc_time            last_seen is_weekend country_code
##    <chr>           <dttm>                  <int> <chr>      <chr>
##  1 171.182.200.160 2018-02-15 14:27:45        NA FALSE      GB
##  2 203.45.67.89    2018-05-20 09:48:30        93 TRUE       AU
##  3 10.0.0.23       2018-08-10 18:12:55        82 FALSE      <NA>
##  4 172.16.32.55    2018-10-05 22:36:17        56 FALSE      <NA>
##  5 8.8.8.8         2018-12-07 11:59:02        62 FALSE      US
##  6 198.51.100.1    2019-01-25 07:24:56        48 FALSE      RO
##  7 106.180.120.93  2019-03-14 16:41:38        48 FALSE      JP
##  8 172.31.15.99    2019-06-08 03:55:20        85 TRUE       <NA>
##  9 67.195.44.68    2019-08-17 20:09:45        70 TRUE       US
## 10 104.244.42.1    2019-11-23 13:30:10        97 TRUE       US
```

QQ2.Implement a fizzbuzz function that takes a single number as input. If the number is divisible by 3, the function should return the string "fizz". If the number is divisible by 5, the function should return the string "buzz". If the number is divisible by both 3 and 5, the function should return the string "fizzbuzz". In all other cases, the function should return the input number as a string. Prove that your fizzbuzz function works as expected on all integers from 0 to 15 using a for loop. Hint: a number n is divisible by a number m if the remainder of the division of n by m is 0; you can test this using the "modulo operator" (%% in R): for instance, 5 %% 2 produces 1, since the remainder of the division of 5 by 2 is 1, whereas 6 %% 3 produces 0, since the remainder of the division of 6 by 3 is 0

```
fizzbuzz <- function(n) {
  if (n %% 3 == 0 && n %% 5 == 0) {
    return("fizzbuzz")
  } else if (n %% 3 == 0) {
    return("fizz")
  } else if (n %% 5 == 0) {
    return("buzz")
  } else {
    return(as.character(n))
  }
}
for (i in 0:15) {
  print(fizzbuzz(i))
}
```

```
## [1] "fizzbuzz"
## [1] "1"
## [1] "2"
## [1] "fizz"
## [1] "4"
## [1] "buzz"
## [1] "fizz"
## [1] "7"
## [1] "8"
## [1] "fizz"
## [1] "buzz"
## [1] "11"
## [1] "fizz"
## [1] "13"
## [1] "14"
## [1] "fizzbuzz"
```

QQ3.Using the map_chr function and the function fizzbuzz that you previously authored, write a new function names fizzbuzz2 that can operate on a vector input i.e., instead of accepting a single number as input it accepts a vector of numbers. Once again, prove that fizzbuzz2 works as expected on using the vector 0:15 as input.

```r
fizzbuzz2 <- function(n) {
  return(map_chr(n, fizzbuzz))
}
result <- fizzbuzz2(0:15)
result
```

```
##  [1] "fizzbuzz" "1"        "2"        "fizz"     "4"        "buzz"
##  [7] "fizz"     "7"        "8"        "fizz"     "buzz"     "11"
## [13] "fizz"     "13"       "14"       "fizzbuzz"
```

QQ4.Using what you know about repetition, conditional execution, and functions in R, write a function that calculates the n-th number in the Fibonacci sequence. Your function should be named fibonacci. It should be such that – fibonacci(1) = 0 – fibonacci(2) = 1 – fibonacci(3) = 1 – fibonacci(4) = 2 – fibonacci(5) = 3 – fibonacci(6) = 5 – and so on.

```r
fibonacci <- function(n) {
  if (n <= 0) {
    return("Invalid input. n must be a positive number.")
  } else if (n == 1) {
    return(0)
  } else if (n == 2) {
    return(1)
  } else {
    return(fibonacci(n-1) + fibonacci(n-2))
  }
}
for(n in 1:6) {
  cat("fibonnaci(",n,")\n", sep="")
  cat("[1]",fibonacci(n),"\n")
}
```

```
## fibonnaci(1)
## [1] 0
## fibonnaci(2)
## [1] 1
## fibonnaci(3)
## [1] 1
## fibonnaci(4)
## [1] 2
## fibonnaci(5)
## [1] 3
## fibonnaci(6)
## [1] 5
```

QQ5.Research the Vectorize function (? Vectorize). Use this code to create a "vectorized" version of the fibonacci function that you previously authored: fibonacci_vec <- Vectorize(fibonacci) Verify that your fibonacci_vec function can now operate on vectors: – what is the result of executing fibonacci(1:10)? – what is instead the result of executing fibonacci_vec(1:10)? Note: in R Markdown, you can specify error = TRUE in a chunk of code e.g.,

```r
#what is the result of executing fibonacci(1:10)?
fibonacci_result <- fibonacci(1:10)
```

```
## Error in if (n <= 0) {: the condition has length > 1
```

```r
fibonacci_result
```

```
## Error in eval(expr, envir, enclos): object 'fibonacci_result' not found
```

```r
#what is instead the result of executing fibonacci_vec(1:10)?
fibonacci_vec <- Vectorize(fibonacci)
result_fib_vec <- fibonacci_vec(1:10)
result_fib_vec
```

```
##  [1]  0  1  1  2  3  5  8 13 21 34
```