Homework 4 Daniel Lee

Daniel Lee

October 13, 2016

Problem 1

```
f1 <- function(x = {y <- 1; 2}, y = 0) {
x + y
}
f1()
```

```
## [1] 3
```

This function returns 3. This is an example of lazy evaluation. When `x + y` is evaluated, the `x` is evaluated first. When `x` is evaluated, `x` takes the value of 2 and `y` takes the value of 1. So, the default value of `y` gets overwritten by the value of `y` when `x` is called. Therefore, when `x` is called, `y` takes the value 1.

Contrast this with the following code:

```
f1 <- function(x = {y <- 1; 2}, y = 0) {
y + x
}
f1()
```

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

In this case, the output is 2. The reason is because `y` is evaluated first. So, `y` takes the value of 0. And then `x` is evaluated to 2. Even though `y` takes the value of 1 when `x` is evaluated, we don't call `y` again. So, `y + x` results in 2.

Problem 2

```
f2 <- function(x = z) {
z <- 100
x
}
f2()
```

```
## [1] 100
```

This function returns 100. `x` does not get evaluated until it is called. This is another example of lazy evaluation. When `x` is called, it takes the value of `z`. And `z` is assigned to be 100. In addition to lazy evaluation, this code also illustrates the fact that you can define arguments within the function.

Problem: Infix Function

```
#This function is the opposite of '%in%' function
#This function sees if a vector contains the elements in the first argument
#is found in the second argument.
#input: x: vector or NULL; the values to be matched. Long vectors are supported
#input: y: vector or NULL; the values to be matched against. Long vectors are supported
#output: a logical vecotr indicating if there is no match or a match for its left operand
#output: If there is a match, function returns FALSE.
#output: If there is no match, the function returns TRUE.
'%nin%' <- function(x, y){
  !(x %in% y) #not '%in%'
}

a <- c(1, 2, 3, 4, 5)
1 %nin% a
```

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

```
6 %nin% a
```

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

```
c(6, 7) %nin% a
```

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

Problem: Function `random_sum()`

```
# copied code from problem set and saved in R script as "random.R"
source("random.R")
```

```
## x: 7 2 1 6 2 5 4 9 2 9
## [1] 47
## x: 8 2 1 1 8
## [1] 520
```

The problem with the code is that `x` is assigned a vector of 10 100s. The function `random_sum()` reassigns only the first n values of `x` and then takes the sum of all the elements in `x`. Therefore, when `show(random_sum(5))` is called, the `random_sum()` will sum up 5 100s in addition to the 5 random numbers. One way to fix this is to assign `x` within the function `random_sum()` with the code `x <- rep(100, n)`. The modified code is shown below:

```
# clear the workspace
rm(list = ls())
random_sum <- function(n) {
  # sum of n random numbers
  x <- rep(100, n) #changed from original code to be defined inside the function
  x[1:n] <- ceiling(10 * runif(n))
  cat("x:", x[1:n], "\n")
  return(sum(x))
}
show(random_sum(10))
```

```
## x: 5 4 2 2 7 4 10 3 10 5
## [1] 52
```

```
show(random_sum(5))
```

```
## x: 8 6 10 10 7
## [1] 41
```

Problem: Scoping

```
# Code from problem set

# initialize an empty list
myFuns <- vector(mode = "list", length = 3)
for (i in 1:length(myFuns)) {
  myFuns[[i]] <- function() {
    return(i)
  }
}
# First evaluation
for (j in 1:length(myFuns)) {
  print(myFuns[[j]]())
}
```

```
## [1] 3
## [1] 3
## [1] 3
```

```
# Second evaluation
for (i in 1:length(myFuns)) {
  print(myFuns[[i]]())
}
```

```
## [1] 1
## [1] 2
## [1] 3
```

a. Explain what is the result of the first evaluation for loop (with "j"). Why is the result 3 every time?

   The result is 3 for every print of `myFuns[[j]]()`. This is because the three objects stored in `myFuns` vector are functions that return the variable "i". And the variable "i", which ends up with the value 3, is in the global environment. So when the function `myFuns[[j]]` prints "i", it prints the "i" from the global environment.

b. Explain the result of the second `for` loop. In particular, where is the value of "i" in the three `MyFuns` functions being found?

   The second `for` loop gives the output 1, 2, 3, because this time, the value of "i" is being reassigned as 1, 2, and 3 by the `for` loop. The value of "i" in the three `MyFuns` functions are being found in the for loop that assigns "i" to 1, 2, and 3.

c. Now consider the following code where the three functions are generated within another function. Why do both loops now give the same result and where is "i" being found?

```
# Code from problem set
funGenerator <- function(len) {
  f <- vector(mode = "list", length = len)
  for (i in seq_len(len)) {
    f[[i]] <- function() {
      i
    }
  }
  return(f)
}

myFuns <- funGenerator(3)
# Third evaluation
for (j in 1:length(myFuns)) {
  print(myFuns[[j]]())
}
```

```
## [1] 3
## [1] 3
## [1] 3
```

```
# Fourth evaluation
for (i in 1:length(myFuns)) {
  print(myFuns[[i]]())
}
```

```
## [1] 3
## [1] 3
## [1] 3
```

Both loops give the same result because the two loops are calling "i" from the `funGenerator` function. In the `funGenerator` function, the value of "i" is assigned 1 to "len" in the `for` loop. And each element in the `f` vector is assigned a function with an unassigned object "i". So, when `myFuns[[i]]()` or `myFuns[[j]]()` is called, the unassigned object "i" in the function looks for an "i" value that is one level up from where the function is being defined. And in the level immediately above where `f[[i]]` is defined, "i" is assigned as 3. That is why both `myFuns[[j]]()` and `myFuns[[i]]()` returns the value 3. In the fourth evaluation, even though the value "i" changes, the "i" that `myFuns[[i]]()` uses is the "i" that is inside the `funGenerator` function.

Transforming the Raw Data

```
library(XML)
library(stringr)

# download file from website
website = "https://www.eia.gov/dnav/pet/hist/LeafHandler.ashx?n=
PET&s=EMM_EPMR_PTE_SCA_DPG&f=W"
download.file(website, "gas-prices.html")

# read in tables with readHTMLTable()
# set which = 5 because the fifth table from the website contains the desired information
dat <- readHTMLTable("gas-prices.html", which = 5,
                      stringsAsFactors = FALSE)

# look at content of 'dat'
head(dat)
```

```
##              V1      V2         V3      V4         V5      V6         V7
## 1    Year-Month Week 1     Week 2 Week 3     Week 4 Week 5        <NA>
## 2      End Date   Value   End Date   Value   End Date   Value   End Date
## 3 Â Â 2000-May       Â       Â Â Â       Â       Â Â Â       Â      Â Â Â
## 4 Â Â 2000-Jun 06/05Â 1.613Â Â Â 06/12Â 1.614Â Â Â 06/19Â 1.618Â Â Â
## 5 Â Â 2000-Jul 07/03Â 1.710Â Â Â 07/10Â 1.729Â Â Â 07/17Â 1.720Â Â Â
## 6 Â Â 2000-Aug 08/07Â 1.666Â Â Â 08/14Â 1.668Â Â Â 08/21Â 1.667Â Â Â
##       V8         V9    V10         V11
## 1   <NA>       <NA>   <NA>        <NA>
## 2  Value   End Date  Value        <NA>
## 3 05/22Â 1.634Â Â Â 05/29Â 1.626Â Â Â
## 4 06/26Â 1.641Â Â Â       Â      Â Â Â
## 5 07/24Â 1.712Â Â Â 07/31Â 1.681Â Â Â
## 6 08/28Â 1.702Â Â Â       Â      Â Â Â
```

```
str(dat)
```

```
## 'data.frame':    220 obs. of  11 variables:
##  $ V1 : chr  "Year-Month" "End Date" "Â Â 2000-May" "Â Â 2000-Jun" ...
##  $ V2 : chr  "Week 1" "Value" "Â" "06/05Â" ...
##  $ V3 : chr  "Week 2" "End Date" "Â Â Â" "1.613Â Â Â" ...
##  $ V4 : chr  "Week 3" "Value" "Â" "06/12Â" ...
##  $ V5 : chr  "Week 4" "End Date" "Â Â Â" "1.614Â Â Â" ...
##  $ V6 : chr  "Week 5" "Value" "Â" "06/19Â" ...
##  $ V7 : chr  NA "End Date" "Â Â Â" "1.618Â Â Â" ...
##  $ V8 : chr  NA "Value" "05/22Â" "06/26Â" ...
##  $ V9 : chr  NA "End Date" "1.634Â Â Â" "1.641Â Â Â" ...
##  $ V10: chr  NA "Value" "05/29Â" "Â" ...
##  $ V11: chr  NA NA "1.626Â Â Â" "Â Â Â" ...
```

```
# select only year 2015 gas prices using grepl() and subset()
dat_2015 <- subset(dat, grepl("2015", dat$V1))
```

EXTRACT DATES

```
############### DATES BASIC FOR LOOP ###############
# This function uses two for loops (one for loop nested in another)
# to extract the dates from table
# This function uses str_extract() to match the regex pattern for dates [0-9]{2}\\/[0-9]{2}
# input: data file
# output: vector of dates
extract_dates_basic_for_loop <- function(data){
  dates_vector <- c()
  for(i in 1:nrow(data)){
    for(j in 1:(ncol(data)/2)){
      dates_vector <- c(dates_vector,
                        str_extract(data[i, j * 2],
                                    "[0-9]{2}\\/[0-9]{2}"))
    }
  }
  return(dates_vector[!is.na(dates_vector)])
}

# vector of dates using basic for loops
extract_dates_basic_for_loop(dat_2015)
```

```
##  [1] "01/05" "01/12" "01/19" "01/26" "02/02" "02/09" "02/16" "02/23"
##  [9] "03/02" "03/09" "03/16" "03/23" "03/30" "04/06" "04/13" "04/20"
## [17] "04/27" "05/04" "05/11" "05/18" "05/25" "06/01" "06/08" "06/15"
## [25] "06/22" "06/29" "07/06" "07/13" "07/20" "07/27" "08/03" "08/10"
## [33] "08/17" "08/24" "08/31" "09/07" "09/14" "09/21" "09/28" "10/05"
## [41] "10/12" "10/19" "10/26" "11/02" "11/09" "11/16" "11/23" "11/30"
## [49] "12/07" "12/14" "12/21" "12/28"
```

```
############### DATES PROFILING ###############
Rprof("extract_dates", memory.profiling = TRUE)

# perform Rprof on 1000 replications of the function because
# one repetition is too short for significant analysis
t <- replicate(n = 1000, extract_dates_basic_for_loop(dat_2015))

Rprof(NULL)

summaryRprof("extract_dates")
```

```
## $by.self
##                                self.time self.pct total.time total.pct
## ".Call"                             1.84    34.33       1.84     34.33
## "[.data.frame"                      0.90    16.79       1.48     27.61
## "stri_extract_first_regex"          0.42     7.84       4.26     79.48
## "type"                              0.38     7.09       0.58     10.82
## "identical"                         0.26     4.85       0.26      4.85
## "["                                 0.22     4.10       1.70     31.72
## "str_extract"                       0.18     3.36       5.02     93.66
## "length"                            0.16     2.99       0.16      2.99
## "match"                             0.14      2.61      0.22      4.10
## "extract_dates_basic_for_loop"      0.12      2.24      5.32     99.25
## "opts"                              0.10      1.87      0.28      5.22
## "type.character"                    0.08      1.49      0.20      3.73
## "names"                             0.08      1.49      0.08      1.49
## "sys.call"                          0.08      1.49      0.08      1.49
## "%in%"                              0.06      1.12      0.28      5.22
## "c"                                 0.06      1.12      0.06      1.12
## "!"                                 0.04      0.75      0.04      0.75
## ".subset2"                          0.04      0.75      0.04      0.75
## "attr"                              0.04      0.75      0.04      0.75
## "FUN"                               0.02      0.37      5.34     99.63
## "ncol"                              0.02      0.37      0.06      1.12
## "dim"                               0.02      0.37      0.04      0.75
## "/"                                 0.02      0.37      0.02      0.37
## ":"                                 0.02      0.37      0.02      0.37
## "all"                               0.02      0.37      0.02      0.37
## "match.fun"                         0.02      0.37      0.02      0.37
## "nargs"                             0.02      0.37      0.02      0.37
##
## $by.total
##                                total.time total.pct self.time self.pct
## "block_exec"                         5.36    100.00      0.00     0.00
## "call_block"                         5.36    100.00      0.00     0.00
## "evaluate"                           5.36    100.00      0.00     0.00
## "evaluate_call"                      5.36    100.00      0.00     0.00
## "in_dir"                             5.36    100.00      0.00     0.00
## "knitr::knit"                        5.36    100.00      0.00     0.00
## "lapply"                             5.36    100.00      0.00     0.00
## "process_file"                       5.36    100.00      0.00     0.00
## "process_group"                      5.36    100.00      0.00     0.00
## "process_group.block"                5.36    100.00      0.00     0.00
## "rmarkdown::render"                  5.36    100.00      0.00     0.00
## "withCallingHandlers"                5.36    100.00      0.00     0.00
## "FUN"                                5.34     99.63      0.02     0.37
## "eval"                               5.34     99.63      0.00     0.00
## "handle"                             5.34     99.63      0.00     0.00
## "replicate"                          5.34     99.63      0.00     0.00
## "sapply"                             5.34     99.63      0.00     0.00
## "withVisible"                        5.34     99.63      0.00     0.00
## "extract_dates_basic_for_loop"       5.32     99.25      0.12     2.24
## "str_extract"                        5.02     93.66      0.18     3.36
## "stri_extract_first_regex"           4.26     79.48      0.42     7.84
## ".Call"                              1.84     34.33      1.84    34.33
## "["                                  1.70     31.72      0.22     4.10
## "[.data.frame"                       1.48     27.61      0.90    16.79
## "type"                               0.58     10.82      0.38     7.09
## "opts"                               0.28      5.22      0.10     1.87
## "%in%"                               0.28      5.22      0.06     1.12
## "identical"                          0.26      4.85      0.26     4.85
## "match"                              0.22      4.10      0.14     2.61
## "type.character"                     0.20      3.73      0.08     1.49
## "length"                             0.16      2.99      0.16     2.99
## "names"                              0.08      1.49      0.08     1.49
## "sys.call"                          0.08      1.49      0.08      1.49
## "c"                                 0.06      1.12      0.06      1.12
## "ncol"                              0.06      1.12      0.02      0.37
## "!"                                 0.04      0.75      0.04      0.75
## ".subset2"                          0.04      0.75      0.04      0.75
## "attr"                              0.04      0.75      0.04      0.75
## "dim"                               0.04      0.75      0.02      0.37
## "/"                                 0.02      0.37      0.02      0.37
## ":"                                 0.02      0.37      0.02      0.37
## "all"                               0.02      0.37      0.02      0.37
## "match.fun"                         0.02      0.37      0.02      0.37
## "nargs"                             0.02      0.37      0.02      0.37
## "dim.data.frame"                    0.02      0.37      0.00      0.00
## "handle_output"                     0.02      0.37      0.00      0.00
## "is.empty"                          0.02      0.37      0.00      0.00
## "plot_calls"                        0.02      0.37      0.00      0.00
## "plot_snapshot"                     0.02      0.37      0.00      0.00
## "w$get_new"                         0.02      0.37      0.00      0.00
##
## $sample.interval
## [1] 0.02
##
## $sampling.time
## [1] 5.36
```

The results of `Rprof()` shows that `.Call` and `[.data.frame` takes the most amount of time to evaluate. I decide to improve the code by using one `for` loop instead of using two.

```
################ DATES BETTER FOR LOOP ################
# This function uses one for loop
# to extract the dates from table.
# This function uses str_extract() to match the regex pattern for dates [0-9]{2}\\/[0-9]{2}
# input: data file
# output: vector of dates
extract_dates_better_for_loop <- function(data){
  dates_vector <- c()
  for(i in 1:nrow(data)){
    dates_vector <- c(dates_vector,
                      str_extract(data[i, ], "[0-9]{2}\\/[0-9]{2}"))
  }
  return(dates_vector[!is.na(dates_vector)])
}

# vector of dates using better for loop
extract_dates_better_for_loop(dat_2015)
```

```
##  [1] "01/05" "01/12" "01/19" "01/26" "02/02" "02/09" "02/16" "02/23"
##  [9] "03/02" "03/09" "03/16" "03/23" "03/30" "04/06" "04/13" "04/20"
## [17] "04/27" "05/04" "05/11" "05/18" "05/25" "06/01" "06/08" "06/15"
## [25] "06/22" "06/29" "07/06" "07/13" "07/20" "07/27" "08/03" "08/10"
## [33] "08/17" "08/24" "08/31" "09/07" "09/14" "09/21" "09/28" "10/05"
## [41] "10/12" "10/19" "10/26" "11/02" "11/09" "11/16" "11/23" "11/30"
## [49] "12/07" "12/14" "12/21" "12/28"
```

```
################ DATES EXTRA CREDIT ################
################ DATES NO FOR LOOP ################
# This function uses vectorization to extract the dates from table
# This function does not use any for loops.
# This function uses str_extract() to match the regex pattern for dates [0-9]{2}\\/[0-9]{2}
# input: data file
# output: vector of dates
extract_dates_no_for_loop <- function(data) {
  dates_vector <- str_extract(c(t(data)), "[0-9]{2}\\/[0-9]{2}")
  return(dates_vector[!is.na(dates_vector)])
}

# vector of dates using no for loops
extract_dates_no_for_loop(dat_2015)
```

```
##  [1] "01/05" "01/12" "01/19" "01/26" "02/02" "02/09" "02/16" "02/23"
##  [9] "03/02" "03/09" "03/16" "03/23" "03/30" "04/06" "04/13" "04/20"
## [17] "04/27" "05/04" "05/11" "05/18" "05/25" "06/01" "06/08" "06/15"
## [25] "06/22" "06/29" "07/06" "07/13" "07/20" "07/27" "08/03" "08/10"
## [33] "08/17" "08/24" "08/31" "09/07" "09/14" "09/21" "09/28" "10/05"
## [41] "10/12" "10/19" "10/26" "11/02" "11/09" "11/16" "11/23" "11/30"
## [49] "12/07" "12/14" "12/21" "12/28"
```

```
################ DATES SYSTEM.TIME() ################
system.time(replicate(n = 1000, extract_dates_basic_for_loop(dat_2015)))
```

```
##    user  system elapsed
##    4.99    0.03    5.78
```

```
system.time(replicate(n = 1000, extract_dates_better_for_loop(dat_2015)))
```

```
##    user  system elapsed
##    3.06    0.00    3.28
```

```
system.time(replicate(n = 1000, extract_dates_no_for_loop(dat_2015)))
```

```
##    user  system elapsed
##    0.39    0.00    0.44
```

The time spent to run the functions decreases from `extract_dates_basic_for_loop(dat_2015)` to `extract_dates_better_for_loop(dat_2015)` to `extract_dates_no_for_loop(dat_2015)`.

EXTRACT PRICES

```
################ PRICES BASIC FOR LOOP ################
# This function uses two for loops (one for loop nested in another)
# to extract the prices from table
# This function uses str_extract() to match the regex pattern for prices [0-9]\\.[0-9]{3}
```

```r
# input: data file
# output: vector of prices
extract_prices_basic_for_loop <- function(data){
  prices_vector <- c()
  for(i in 1:nrow(data)){
    for(j in 1:(ncol(data)/2)){
      prices_vector <- c(prices_vector,
                         str_extract(data[i, j*2+1],
                                     "[0-9]\\.[0-9]{3}"))
    }
  }
  return(as.numeric(prices_vector[!is.na(prices_vector)]))
}

# vector of prices using basic for loops
extract_prices_basic_for_loop(dat_2015)
```

```
##  [1] 2.671 2.594 2.484 2.440 2.441 2.627 2.798 2.959 3.418 3.439 3.356
## [12] 3.267 3.209 3.147 3.102 3.158 3.433 3.711 3.732 3.807 3.757 3.693
## [23] 3.591 3.511 3.480 3.450 3.432 3.880 3.897 3.812 3.724 3.565 3.584
## [34] 3.483 3.342 3.266 3.155 3.072 2.994 2.949 2.914 2.861 2.847 2.817
## [45] 2.824 2.780 2.716 2.691 2.679 2.654 2.736 2.825
```

```r
############### PRICES PROFILING ###############
Rprof("extract_prices", memory.profiling = TRUE)

# perform Rprof on 1000 replications of the function because
# one repetition is too short for significant analysis
t <- replicate(n = 1000, extract_prices_basic_for_loop(dat_2015))

Rprof(NULL)
summaryRprof("extract_prices")
```

```
## $by.self
##                                 self.time self.pct total.time total.pct
## ".Call"                              1.62    29.67       1.62     29.67
## "[.data.frame"                       1.12    20.51       1.92     35.16
## "type"                               0.34     6.23       0.60     10.99
## "identical"                          0.30     5.49       0.30      5.49
## "str_extract"                        0.22     4.03       5.06     92.67
## "stri_extract_first_regex"           0.22     4.03       4.24     77.66
## "extract_prices_basic_for_loop"      0.20     3.66       5.46    100.00
## "match"                              0.16     2.93       0.34      6.23
## "["                                  0.14     2.56       2.06     37.73
## "names"                              0.14     2.56       0.14      2.56
## "opts"                               0.12     2.20       0.34      6.23
## "type.character"                     0.12     2.20       0.26      4.76
## ".subset"                            0.10     1.83       0.10      1.83
## "length"                             0.10     1.83       0.10      1.83
## "sys.call"                           0.10     1.83       0.10      1.83
## "%in%"                               0.08     1.47       0.42      7.69
## "c"                                  0.08     1.47       0.08      1.47
## "dim"                                0.06     1.10       0.14      2.56
## "attr"                              0.06     1.10       0.06      1.10
## ".row_names_info"                   0.04     0.73       0.04      0.73
## "all"                               0.04     0.73       0.04      0.73
## "nargs"                             0.04     0.73       0.04      0.73
## ".subset2"                          0.02     0.37       0.02      0.37
## "+"                                 0.02     0.37       0.02      0.37
## "anyNA"                             0.02     0.37       0.02      0.37
##
## $by.total
##                                  total.time total.pct self.time self.pct
## "extract_prices_basic_for_loop"        5.46    100.00      0.20     3.66
## "block_exec"                           5.46    100.00      0.00     0.00
## "call_block"                           5.46    100.00      0.00     0.00
## "eval"                                 5.46    100.00      0.00     0.00
## "evaluate"                             5.46    100.00      0.00     0.00
## "evaluate_call"                        5.46    100.00      0.00     0.00
## "FUN"                                  5.46    100.00      0.00     0.00
## "handle"                               5.46    100.00      0.00     0.00
## "in_dir"                               5.46    100.00      0.00     0.00
## "knitr::knit"                          5.46    100.00      0.00     0.00
## "lapply"                               5.46    100.00      0.00     0.00
## "process_file"                         5.46    100.00      0.00     0.00
## "process_group"                        5.46    100.00      0.00     0.00
## "process_group.block"                  5.46    100.00      0.00     0.00
## "replicate"                            5.46    100.00      0.00     0.00
## "rmarkdown::render"                    5.46    100.00      0.00     0.00
## "sapply"                               5.46    100.00      0.00     0.00
## "withCallingHandlers"                  5.46    100.00      0.00     0.00
## "withVisible"                          5.46    100.00      0.00     0.00
## "str_extract"                          5.06     92.67      0.22     4.03
## "stri_extract_first_regex"             4.24     77.66      0.22     4.03
## "["                                    2.06     37.73      0.14     2.56
## "[.data.frame"                         1.92     35.16      1.12    20.51
## ".Call"                                1.62     29.67      1.62    29.67
## "type"                                 0.60     10.99      0.34     6.23
## "%in%"                                 0.42      7.69      0.08     1.47
## "match"                                0.34      6.23      0.16     2.93
## "opts"                                 0.34      6.23      0.12     2.20
## "identical"                            0.30      5.49      0.30     5.49
## "type.character"                       0.26      4.76      0.12     2.20
## "names"                                0.14      2.56      0.14     2.56
## "dim"                                  0.14      2.56      0.06     1.10
## "ncol"                                 0.14      2.56      0.00     0.00
## ".subset"                              0.10      1.83      0.10     1.83
## "length"                               0.10      1.83      0.10     1.83
## "sys.call"                             0.10      1.83      0.10     1.83
## "c"                                    0.08      1.47      0.08     1.47
## "dim.data.frame"                       0.08      1.47      0.00     0.00
## "attr"                                 0.06      1.10      0.06     1.10
## ".row_names_info"                      0.04      0.73      0.04     0.73
## "all"                                  0.04      0.73      0.04     0.73
## "nargs"                                0.04      0.73      0.04     0.73
## ".subset2"                             0.02      0.37      0.02     0.37
## "+"                                    0.02      0.37      0.02     0.37
## "anyNA"                              0.02      0.37      0.02     0.37
##
## $sample.interval
## [1] 0.02
##
## $sampling.time
## [1] 5.46
```

The results of `Rprof()` shows that `.Call` and `[.data.frame` takes the most amount of time to evaluate. I decide to improve the code by using one `for` loop instead of using two.

```
################ PRICES BETTER FOR LOOP ################
# This function uses one for loop
# to extract the prices from table
# This function uses str_extract() to match the regex pattern for prices [0-9]\\.[0-9]{3}
# input: data file
# output: vector of prices
extract_prices_better_for_loop <- function(data){
  prices_vector <- c()
  for(i in 1:nrow(data)){
    prices_vector <- c(prices_vector,
                       str_extract(data[i, ], "[0-9]\\.[0-9]{3}"))
  }
  return(as.numeric(prices_vector[!is.na(prices_vector)]))
}

# vector of prices using better for loop
extract_prices_better_for_loop(dat_2015)
```

```
##  [1] 2.671 2.594 2.484 2.440 2.441 2.627 2.798 2.959 3.418 3.439 3.356
## [12] 3.267 3.209 3.147 3.102 3.158 3.433 3.711 3.732 3.807 3.757 3.693
## [23] 3.591 3.511 3.480 3.450 3.432 3.880 3.897 3.812 3.724 3.565 3.584
## [34] 3.483 3.342 3.266 3.155 3.072 2.994 2.949 2.914 2.861 2.847 2.817
## [45] 2.824 2.780 2.716 2.691 2.679 2.654 2.736 2.825
```

```
################ PRICES EXTRA CREDIT ################
################ PRICES NO FOR LOOP ################
# This function uses vectorization to extract the prices from table
# This function does not use any for loops.
# This function uses str_extract() to match the regex pattern for prices [0-9]\\.[0-9]{3}
# input: data file
# output: vector of prices
extract_prices_no_for_loop <- function(data) {
  prices_vector <- str_extract(c(t(data)), "[0-9]\\.[0-9]{3}")
  return(as.numeric(prices_vector[!is.na(prices_vector)]))
}

# vector of prices using no for loops
extract_prices_no_for_loop(dat_2015)
```

```
##  [1] 2.671 2.594 2.484 2.440 2.441 2.627 2.798 2.959 3.418 3.439 3.356
## [12] 3.267 3.209 3.147 3.102 3.158 3.433 3.711 3.732 3.807 3.757 3.693
## [23] 3.591 3.511 3.480 3.450 3.432 3.880 3.897 3.812 3.724 3.565 3.584
## [34] 3.483 3.342 3.266 3.155 3.072 2.994 2.949 2.914 2.861 2.847 2.817
## [45] 2.824 2.780 2.716 2.691 2.679 2.654 2.736 2.825
```

```
############### PRICES SYSTEM.TIME() ###############
system.time(replicate(n = 1000, extract_prices_basic_for_loop(dat_2015)))
```

```
##    user  system elapsed
##    4.93    0.02    5.21
```

```
system.time(replicate(n = 1000, extract_prices_better_for_loop(dat_2015)))
```

```
##    user  system elapsed
##    3.02    0.01    3.22
```

```
system.time(replicate(n = 1000, extract_prices_no_for_loop(dat_2015)))
```

```
##    user  system elapsed
##    0.42    0.00    0.43
```

The time spent to run the functions decreases from `extract_prices_basic_for_loop(dat_2015)` to `extract_prices_better_for_loop(dat_2015)` to `extract_prices_no_for_loop(dat_2015)`.

Export 'clean-gas-prices-2015.csv'

```
dates <- extract_dates_no_for_loop(dat_2015)
prices <- extract_prices_no_for_loop(dat_2015)

# create data.frame with columns "week", "date", "price"
clean_gas_prices_2015 <- data.frame(week = 1:length(dates),
                                    date = dates,
                                    price = prices)

# examine the content of 'clean_gas_prices_2015'
head(clean_gas_prices_2015)
```

```
##   week  date price
## 1    1 01/05 2.671
## 2    2 01/12 2.594
## 3    3 01/19 2.484
## 4    4 01/26 2.440
## 5    5 02/02 2.441
## 6    6 02/09 2.627
```

```
# export 'clean-gas-prices-2015.csv' using write.csv()
write.csv(clean_gas_prices_2015, "clean-gas-prices-2015.csv", row.names = FALSE)
```