STATS506-Problem Set 5

Shenyi Tang

2024-11-19

Shenyi Tang's GitHub Repo For STATS 506 FA 2024

https://github.com/shenyi-tang/stats506-computing-methods-and-tools.git

```
library(roxygen2)
library(Rcpp)
library(ggplot2)
library(tidyverse)
library(plotly)
```

Problem 1 - OOP Programming

a. **For the `rational` class, define the following:**

1. **A constructor**
2. **A validator that ensures the denominator is non-zero**
3. **A `show` method**
4. **A `simplify` method, to obtain the simplest form (e.g. `simplify(2/4)` produces `1/2`)**
5. **A `quotient` method (e.g. `quotient(3/7)` produces `.42857143...`). It should support a `digits` argument *but only in the printing, not the return result* (Hint: what does `print` return?).**
6. **Addition, subtraction, multiplication, division. These should all return a `rational`.**
7. **You'll (probably) need GCD and LCM as part of some of these calculations; include these functions using Rcpp. Even if you don't need these functions for another calculation, include them.**

```
# a.1 define the rational class
##' @title define "rational" class
##' @slot numerator, an integer, the number above the line in the common fraction
##' @slot denominator, an integer, the number below the line in the common fraction
##'
rational <- setClass("rational",
                     slots = c(numerator = "numeric",
                               denominator = "numeric"))
```

```r
# a.2 a validator
##' @title set a validator for 'rational' class
##' @detail to check whether both the numerator and denominator are integers
##' @detail to check whether the denominator is non-zero
##'
setValidity("rational", function(object){
  if (object@denominator == 0) {
    stop(paste0("@denominator = 0, is not a valid denominator"))
  }
  else if(!is.numeric(object@numerator) || !is.numeric(object@denominator)) {
    stop("Both the numerator and denominator should be numeric")
  }
  else if (floor(object@numerator) != object@numerator ||
   ↪  !is.numeric(object@numerator)) {
    stop(paste0("@numerator = ", object@numerator, " is not a valid numerator"))
  }
  else if (floor(object@denominator) != object@denominator ||
   ↪  !is.numeric(object@denominator)) {
    stop(paste0("@denominator = ", object@denominator, " is not a valid denominator"))
  }
  return(TRUE)
})

# a.3 a show method
##' @title show the rational in a fraction style
##'
setMethod("show", "rational",
          function(object){
            cat(object@numerator, "/", object@denominator, "\n", sep = "")
          })

# a.4 a simplify method
# GCD - Greatest Common Divisor
# LCM - Least Common Multiple
cppFunction("
  int gcd(int a, int b){
    while(b != 0){
      int temp = b;
      b = a % b;
      a = temp;
    }
    return a;
  }

  int lcm(int a, int b){
    return a * b / gcd(a, b);
  }
            ")
```

```r
##' @title create a method to simplify the fraction
##' @param object, with a "rational" class
##' @detail divide the numerator and denominator respectively by their GCD
##'
setGeneric("simplify",
           function(object){
             standardGeneric("simplify")
           })
```

```
Creating a new generic function for 'simplify' in the global environment
```

```r
setMethod("simplify", "rational",
          function(object){
            g <- gcd(abs(object@numerator), abs(object@denominator))
            simplified_numerator <- object@numerator / g
            simplified_denominator <- object@denominator / g
            rational(numerator = simplified_numerator,
                     denominator = simplified_denominator)
          })

# a.5 a quotient method
##' @title calculate the quotient of the fraction
##' @param object, with a "rational" class
##' @param digits, default value is null
##'
setGeneric("quotient",
           function(object, digits = NULL){
             standardGeneric("quotient")
           })

setMethod("quotient", "rational",
          function(object, digits = NULL){
            q <- as.numeric(object@numerator) / as.numeric(object@denominator)
            if (!is.null(digits)) {
              if (!is.numeric(digits) || digits != floor(digits)){
                stop("Digits should be an integer")
              }
              # use "format" to control the output of the result
              formatted_q <- format(round(q, digits), nsmall = digits, scientific =
 ↪  FALSE)
              print(formatted_q)
            }
            else {
              print(format(q, scientific = FALSE))
            }
            return(invisible(q))
          })

# a.6 Operations: Addition, subtraction, multiplication, and division
```

```
# Addition
setMethod("+", signature(e1 = "rational",
                         e2 = "rational"),
          function(e1, e2){
            new_numerator <- e1@numerator * e2@denominator + e2@numerator *
 ↪  e1@denominator
            new_denominator <- e1@denominator * e2@denominator
            simplify(rational(numerator = new_numerator,
                              denominator = new_denominator))
          })

# subtraction
setMethod("-", signature(e1 = "rational",
                         e2 = "rational"),
          function(e1, e2){
            new_numerator <- e1@numerator * e2@denominator - e2@numerator *
 ↪  e1@denominator
            new_denominator <- e1@denominator * e2@denominator
            simplify(rational(numerator = new_numerator,
                              denominator = new_denominator))
          })

# multiplication
setMethod("*", signature(e1 = "rational",
                         e2 = "rational"),
          function(e1, e2){
            new_numerator <- e1@numerator * e2@numerator
            new_denominator <- e1@denominator * e2@denominator
            simplify(rational(numerator = new_numerator,
                              denominator = new_denominator))
          })

# division
setMethod("/", signature(e1 = "rational",
                         e2 = "rational"),
          function(e1, e2){
            if (e2@numerator == 0){
              stop("Divisor cannot be zero")
            }
            new_numerator <- e1@numerator * e2@denominator
            new_denominator <- e1@denominator * e2@numerator
            simplify(rational(numerator = new_numerator,
                              denominator = new_denominator))
          })
```

b. Use your `rational` class to create 3 objects

1. `r1`: $\frac{24}{6}$

2. r2: $\frac{7}{230}$
3. r3: $\frac{0}{4}$

```
# create 3 rational objects
r1 <- rational(numerator = 24, denominator = 6)
r2 <- rational(numerator = 7, denominator = 230)
r3 <- rational(numerator = 0, denominator = 4)

# test cases
r1
```

```
24/6
```

```
r3
```

```
0/4
```

```
r1 + r2
```

```
927/230
```

```
r1 - r2
```

```
913/230
```

```
r1 * r2
```

```
14/115
```

```
r1 / r2
```

```
920/7
```

```
r1 + r3
```

```
4/1
```

```
r1 * r3
```

```
0/1
```

```
r2 / r3
```

```
Error in r2/r3: Divisor cannot be zero
```

```
quotient(r1)
```

```
[1] "4"
```

```
quotient(r2)
```

```
[1] "0.03043478"
```

```
quotient(r2, digits = 3.14)
```

```
Error in quotient(r2, digits = 3.14): Digits should be an integer
```

```
quotient(r2, digits = "avocado")
```

```
Error in quotient(r2, digits = "avocado"): Digits should be an integer
```

```
q2 <- quotient(r2, digits = 3)
```

```
[1] "0.030"
```

```
q2
```

```
[1] 0.03043478
```

```
quotient(r3)
```

```
[1] "0"
```

```
simplify(r1)
```

```
4/1
```

```
simplify(r2)
```

```
7/230
```

```
simplify(r3)
```

```
0/1
```

c. Show that your validator does not allow the creation of rational's with 0 denominator, and check other malformed input to your constructor.

```
rational(numerator = 2, denominator = 0)
```

```
Error in validityMethod(object): @denominator = 0, is not a valid denominator
```

```
rational(numerator = "n", denominator = 2)
```

```
Error in validObject(.Object): invalid class "rational" object: invalid object for slot
"numerator" in class "rational": got class "character", should be or extend class
"numeric"
```

```
rational(numerator = "3", denominator = 4)
```

```
Error in validObject(.Object): invalid class "rational" object: invalid object for slot
"numerator" in class "rational": got class "character", should be or extend class
"numeric"
```

Problem 2 - plotly

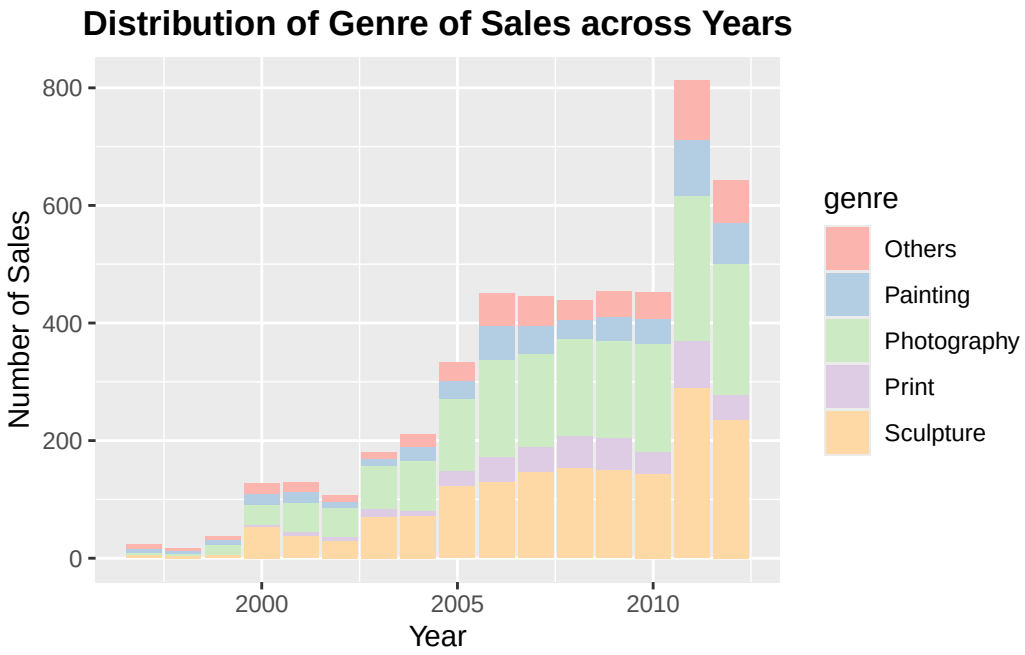
```
df <- read.csv("df_for_ml_improved_new_market.csv")
```

a. Regenerate your plot which addresses the second question from last time:

ii Does the distribution of genre of sales across years appear to change?

```
# transfer the origin data set to longer table
# remove the prefix "Genre___"
pivot_df <- df %>%
  tidyr::pivot_longer(
    cols = c(Genre___Photography, Genre___Print, Genre___Sculpture, Genre___Painting,
    ↪  Genre___Others),
    names_to = 'genre',
    values_to = 'if.genre'
  ) %>%
  filter(if.genre == 1) %>%
  mutate(genre = gsub(".*___", "", genre))


pivot_df %>%
  group_by(year, genre) %>%
  summarise(count = n()) %>%
  ggplot(aes(x = year, y = count, fill = genre)) +
  geom_bar(stat = "identity", position = "stack") +
  labs(title = "Distribution of Genre of Sales across Years",
       x = "Year",
       y = "Number of Sales") +
  theme(plot.title = element_text(hjust = 0.5,
                                  face = "bold")) +
  scale_fill_brewer(palette = "Pastel1")
```



b. Generate an interactive plot with plotly that can address both of these questions from last time:

i Is there a change in the sales price in USD over time?

ii How does the genre affect the change in sales price over time?

Problem 3 - data.table

a.