Homework 1

Ze Yang (zey@andrew.cmu.edu)

Due Tuesday, September 5 at 5:30 PM

You should submit the Rmd file with your answers in the appropriate spaces. Rename the file as `YOURANDREWID_HW1.Rmd` and submit it via Canvas. You should also "Knit" the `.Rmd` file and submit the resulting file `YOURANDREWID_HW1.pdf` as well.

Of course, any code that is written should be tested. In each case you are requested to include some examples showing that your code works. Think about how your example(s) can illustrate the range of possibilities that the code could face.

---

1. Suppose that `frameex` is a data frame. Write a single line of R code that returns a logical vector of length equal to the number of rows in `frameex`. The value should be `TRUE` if the corresponding row is **complete**, i.e., it has no `NA` values. If there is at least one `NA` value in the row, then the value should be `FALSE`.

- **Solution**:

```r
is.complete.row = function(frame) {
  #' Create indicator vactor indicating whether each
  #' row of data frame contains no NA.
  #' @param frame: data.frame object.
  #' @return a logical vector of length equal to \code{nrow(frame)}
  #' \code{TRUE} at i-th position meaning i-th row of frame has no NA.

  # is.na(frame) returns a data.frame of the same shape, with
  # TRUE entry indicating corresponding entry is NA in frame.
  # A row is complete if the corresponding row in is.na(frame) has
  # no TRUE. So the row should sums to zero.
  return(rowSums(is.na(frame)) == 0)
}

# Create some test cases.
test.frame.1 = data.frame()
test.frame.2 = data.frame(
  letter.grades = c(NA, "A", NA, "B", "B", "A"),
  id = c(1, 2, 3, 4, 5, NA)
)
test.frame.3 = data.frame(
  letter.grades = c(NA, "A", NA, "B", "B", "A"),
  grades = c(0, 90, NA, 85, 87, Inf)
)
```

```
is.complete.row(test.frame.1)
```

```
## logical(0)
```

```
is.complete.row(test.frame.2)
```

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

```
# Inf is not treated as NA.
is.complete.row(test.frame.3)
```

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

---

2. Suppose that `x` and `y` are both vectors, and I want to find the first instance of **any** entry in `x` appearing in `y`. Write R code to do this. Be careful about `NA` values. If no element of `x` is in `y`, the value should be `NA`. For example, if `x=c(1,2,3)` and `y=c(4,2,1)`, then the code should return `2` because the second position in `y` is the first to contain an element that is in `x`. **Hint:** One approach is to use `match()` and `sort()`.

- **Solution**:

```
first.match = function(x, y) {
  #' @param x,y: two vectors.
  #' @return The position of the entry in y which is the first
  #' to match some entry in x. Return NA if no entry in y matches
  #' any entry of x.
  return(sort(match(x,y),na.last=TRUE)[1])
}

# create some test cases
x.1 = c(1, 2, 4, 8)
y.1 = c(0, 4, 8)
x.2 = c()
y.2 = c(1, 2)
x.3 = c(1, 2)
y.3 = c()
x.4 = c()
y.4 = c()
x.5 = c(NA, 1, 2)
y.5 = c(7, 8, 9, 10, 1)
x.6 = c(NA, 1, NA)
y.6 = c(NA, NA, 8, 9, 10, 1)
```

```
first.match(x.1, y.1) # expected 2
```

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

```
first.match(x.2, y.2) # expected NA
```

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

```r
first.match(x.3, y.4) # expected NA
```

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

```r
first.match(x.4, y.4) # expected NA
```

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

```r
first.match(x.5, y.5) # expected 5
```

```
## [1] 5
```

```r
first.match(x.6, y.6) # expected 1
```

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

---

3. What does R do in a case when a reference is made to a position that is beyond the range of a vector? For example, if `x=c(1,2,3)`, and reference is made to `x[4]`?

- **Answer**: The reference to a position beyond the range of the vector gets an `NA`.
- **Example**:

```r
# illustrate R's behavior for indices out of bound.
x = c(1,2,3)
x[c(1:10)]
```

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

---

4. Suppose that `n` and `m` are both positive integers. Write a single line of R code that creates a `n` by `m` matrix whose j^{th} column is filled with j.

- **Solution**:

```r
fill.column.matrix = function(n, m) {
  #' Create a n by m matirx
  #' row of data frame contains no NA.
  #' @param n: number of rows.
  #' @param m: number of columns.
  #' @return the required matrix.

  # Slightly modify the "fill.row" we discussed in class
  # by replacing rep(..., times) with rep(..., each).
  #
  # This approach has a flaw: when m=0, n=1, it returns
  # [[1, 0]] in stead of empty matrix. Since 1:0 is interpreted
  # as a decreasing sequence from 1 to 0.
  #
  # We have to wrap it in a if-else condition to rule out this anomaly.
  return(if(m!=0) matrix(rep(1:m, each=n), nrow=n)
         else matrix(integer(0),nrow=0))
}

fill.column.matrix(0,0)
```

```
## <0 x 0 matrix>
```

```r
fill.column.matrix(0,1)
```

```
## <0 x 0 matrix>
```

```r
# Anomalous case! Expect <0 * 0 matrix> here.
matrix(rep(1:0, each=1), nrow=1)
```

```
##      [,1] [,2]
## [1,]    1    0
```

```r
# Modified version
fill.column.matrix(1,0)
```

```
## <0 x 0 matrix>
```

```r
fill.column.matrix(1,8)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    2    3    4    5    6    7    8
```

```r
fill.column.matrix(8,1)
```

```
##      [,1]
## [1,]    1
## [2,]    1
## [3,]    1
## [4,]    1
## [5,]    1
## [6,]    1
## [7,]    1
## [8,]    1
```

```r
fill.column.matrix(6,5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    1    2    3    4    5
## [3,]    1    2    3    4    5
## [4,]    1    2    3    4    5
## [5,]    1    2    3    4    5
## [6,]    1    2    3    4    5
```

---

5. What does R do in a case where `cbind()` is used in an attempt to join two matrices that have different numbers of rows?

- **Answer**: The code won't compile, `cbind()` throws an exception saying the dimensions are incompatible.
- **Example**:

```
# illustrate cbind's behavior for incompatible dimensions.
A = matrix(1:10, nrow=2)
B = matrix(1:12, nrow=3)
print(A)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

```
print(B)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
tryCatch(cbind(A,B), error=function(e){print(e)})
```

```
## <simpleError in cbind(A, B): number of rows of matrices must match (see arg 2)>
```

---

6. Suppose that `factex` is a vector, and a **factor**. The levels of this factor are `a`, `b`, and `c`. Does it make sense to write `factex >= "b"`? **Be careful!** Explain your response clearly.

- **Answer**: The comparison is meaningful if and only if `factex` is specified as `ordered = T`. If the vector of factors is specified as *unordered*, then the comparison is meaningless. The code will still compile but throw a warning, and comparison results will be `NA`.
- **Example**:

```
factex = factor(c('a', 'b', 'c', 'a', 'b'),
                levels=c('a','b','c'), ordered = T)
factex >= 'b'
```

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

```
factex.unordered = factor(c('a', 'b', 'c', 'a', 'b'),
                          levels=c('a','b','c'), ordered = F)
factex.unordered >= 'b'
```

```
## Warning in Ops.factor(factex.unordered, "b"): '>=' not meaningful for
## factors
```

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

---

7. If you construct a data frame from a matrix, but do not supply names for the variables, what does R do?

- **Answer**: The `data.frame` will generate default names, called `X1, X2, ..., Xm`, through the m-th column of the matrix. These names can be used in the usual way.
- **Example**:

```
test.matrix = matrix(c(1:16), ncol=4)
test.frame.4 = data.frame(test.matrix)
test.frame.4
```

```
##   X1 X2 X3 X4
## 1  1  5  9 13
## 2  2  6 10 14
## 3  3  7 11 15
## 4  4  8 12 16
```

```
str(test.frame.4)
```

```
## 'data.frame':    4 obs. of  4 variables:
##  $ X1: int  1 2 3 4
##  $ X2: int  5 6 7 8
##  $ X3: int  9 10 11 12
##  $ X4: int  13 14 15 16
```

```
test.frame.4$X2
```

```
## [1] 5 6 7 8
```

---

8. Give at least three ways a data frame differs from a matrix.

- `matrix` is more *homogeneous*, all entries must be of the same type. When consturcted with different data types, it tries to coerce them to the same type. Columns of `data.frame` can be of different types.
- Given same data, `matrix` is more memory efficient than `data.frame`.
- `matrix` supports a variety of arithmetic operators that are involved in linear algebra, `data.frame` does not.
- `data.frame` has an embedded list-of-vectors structure, while `matrix` does not. So they have different selecting/subsetting methods.

```
M = cbind(c(1,2,3), c('s','m','b'))
colnames(M) = c('col1', 'col2')
```

---

9. Explore the function `unique()`. How could you use this function to remove redundant **columns** in a matrix or data frame?

- **Solution**:

```
remove.duplicate.cols = function(data) {
  #' Remove the duplicate columns of a 2 dimensional data set.
```

```r
  #' @param data: matrix or data.frame
  #' @return data with its duplicate columns removed.

  if(class(data) == "matrix") {
    return(unique(data, MARGIN=2))
  } else if(class(data) == "data.frame") {
    t(unique(t(data), MARGIN=1))
  } else {
    return(data)
  }
}

# create some test cases
test.matrix.2 = cbind(c(1,2,3,4), c(1,2,3,4), c(2,3,4,5),
                      c(3,5,6,6), c(3,5,6,6))
test.frame.5 = data.frame(
  col1 = c(1,1,3,4),
  col2 = c(2,2,4,5),
  col3 = c(1,1,3,4),
  col4 = c(3,5,6,6),
  col5 = c(2,2,4,5),
  col6 = c(3,5,6,6)
)

test.matrix.2
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    1    2    3    3
## [2,]    2    2    3    5    5
## [3,]    3    3    4    6    6
## [4,]    4    4    5    6    6
```

```r
remove.duplicate.cols(test.matrix.2)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    2    3    5
## [3,]    3    4    6
## [4,]    4    5    6
```

```r
test.frame.5
```

```
##   col1 col2 col3 col4 col5 col6
## 1    1    2    1    3    2    3
## 2    1    2    1    5    2    5
## 3    3    4    3    6    4    6
## 4    4    5    4    6    5    6
```

```r
remove.duplicate.cols(test.frame.5)
```

```
##      col1 col2 col4
## [1,]    1    2    3
## [2,]    1    2    5
## [3,]    3    4    6
## [4,]    4    5    6
```