homework1

Claire Whittington

2023-10-02

R Markdown

1) 1 – 3 pts

Create a vector of three elements (2,4,6) and name that vector vec_a. Create a second vector, vec_b, that contains (8,10,12). Add these two vectors together and name the result vec_c.

```
vec_a = c(2,4,6)
vec_b = c(8,10,12)

vec_c = vec_a + vec_b

vec_c
```

```
## [1] 10 14 18
```

(2) 2 – 5 pts (note there are 3 questions to answer in addition to the coding tasks.)

Create a vector, named vec_d, that contains only two elements (14,20). Add this vector to vec_a. What is the result and what do you think R did (look up the recycling rule using Google)? What is the warning message that R gives you?

```
vec_d = c(14,20)

vec_a = vec_a + vec_d
```

```
## Warning in vec_a + vec_d: longer object length is not a multiple of shorter
## object length
```

```
# Warning: longer object length is not a multiple of shorter object length
# Explanation: The output will be length max(length_x, length_y),
# and a warning will be thrown if the length of the longer vector is not an
# integer multiple of the length of the shorter vector.
```

(3) 3 – 4 pts (note there are 3 questions to answer in addition to the coding task.)

Next add 5 to the vector vec_a. What is the result and what did R do? Why doesn't in give you a warning message similar to what you saw in the previous problem?

```
vec_a = vec_a + 5
vec_a
```

```
## [1] 21 29 25
```

```
# result: [1]  7  9 11
# r added 5 to each number in vec_a
# r likely understands that this means "add 5 to each item in the vector", which
# will not throw a warning
```

(4) 5 – 2 pts

Generate the vector of even numbers {2,4,6,. . . ,20} a) using the seq() function and b) Using the a:b shortcut and some subsequent algebra. Hint: Generate the vector 1-10 and then multiple it by 2.

```
evens_a = seq(2,20,2) # a

evens_b = (1:10) * 2

evens_a
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
evens_b
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

(5) 6 – 1 pt

Generate a vector of 21 elements that are evenly placed between 0 and 1 using the seq() command and name this vector x

```
scale = 1/21

vector = seq(0,1,scale)

vector
```

```
##  [1] 0.00000000 0.04761905 0.09523810 0.14285714 0.19047619 0.23809524
##  [7] 0.28571429 0.33333333 0.38095238 0.42857143 0.47619048 0.52380952
## [13] 0.57142857 0.61904762 0.66666667 0.71428571 0.76190476 0.80952381
## [19] 0.85714286 0.90476190 0.95238095 1.00000000
```

(6) 8 – 1 pt

Generate the vector {2,2,2,2,4,4,4,4,8,8,8,8} using the rep() command. You might need to check the help file for rep() to see all of the options that rep() will accept. In particular, look at the optional argument each=.

```
vector = c(rep(2, 4),rep(4,4),rep(8,4))
vector
```

```
##  [1] 2 2 2 2 4 4 4 4 8 8 8 8
```

(7) 11 – 7 pts

Create and manipulate a data frame.

a) Create a data.frame named my.trees that has the following columns: Girth = {8.3, 8.6, 8.8, 10.5, 10.7, 10.8, 11.0} Height= {70, 65, 63, 72, 81, 83, 66} Volume= {10.3, 10.3, 10.2, 16.4, 18.8, 19.7, 15.6}

```
my.trees = data.frame(
  Girth = c(8.3, 8.6, 8.8, 10.5, 10.7, 10.8, 11.0),
  Height= c(70, 65, 63, 72, 81, 83, 66),
  Volume= c(10.3, 10.3, 10.2, 16.4, 18.8, 19.7, 15.6)
)
my.trees
```

```
##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2
## 4  10.5     72   16.4
## 5  10.7     81   18.8
## 6  10.8     83   19.7
## 7  11.0     66   15.6
```

b) Without using dplyr functions, extract the third observation (i.e. the third row)

```
value = my.trees[3] # r is not zero indexed
value
```

```
##   Volume
## 1   10.3
## 2   10.3
## 3   10.2
## 4   16.4
## 5   18.8
## 6   19.7
## 7   15.6
```

c) Without using dplyr functions, extract the Girth column referring to it by name (don't use whatever order you placed the columns in).

```
girth <- my.trees[, 'Girth']
girth
```

```
## [1]  8.3  8.6  8.8 10.5 10.7 10.8 11.0
```

d) Without using dplyr functions, print out a data frame of all the observations except for the fourth observation. (i.e. Remove the fourth observation/row.)

```
fourth_removed = my.trees[-4, ]
fourth_removed
```

```
##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2
## 5  10.7     81   18.8
## 6  10.8     83   19.7
## 7  11.0     66   15.6
```

e) Without using dplyr functions, use the which() command to create a vector of row indices that have a girth greater than 10. Call that vector index.

```
index <- which(my.trees$Girth > 10)
index
```

```
## [1] 4 5 6 7
```

f) Without using dplyr functions, use the index vector to create a small data set with just the large girth trees.

```
large_girth_trees = data.frame(
                      c(my.trees[index,]))
large_girth_trees
```

```
##   Girth Height Volume
## 1  10.5     72   16.4
## 2  10.7     81   18.8
## 3  10.8     83   19.7
## 4  11.0     66   15.6
```

g) Without using dplyr functions, use the index vector to create a small data set with just the small girth trees.

```
small_girth_trees = data.frame(
                          my.trees[which(my.trees$Girth<10),])
small_girth_trees
```

```
##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2
```

(8) 12 – 2 pts

The following code creates a data.frame and then has two different methods for removing the rows with NA values in the column Grade. Explain the difference between the two

```
df <- data.frame(name= c('Alice','Bob','Charlie','Daniel'),
                 Grade = c(6,8,NA,9))

df[ -which(  is.na(df$Grade) ), ] # removes all data where grades are NA
```

```
##     name Grade
## 1  Alice     6
## 2    Bob     8
## 4 Daniel     9
```

```
df[  which( !is.na(df$Grade) ), ] # shows all data where grades are not NA
```

```
##     name Grade
## 1  Alice     6
## 2    Bob     8
## 4 Daniel     9
```

(9) 14 – 3 pts

Create and manipulate a list.

Create a list named my.test with elements x = c(4,5,6,7,8,9,10) y = c(34,35,41,40,45,47,51) slope = 2.82 p.value = 0.000131

```
my.test <- list(
  x = c(4, 5, 6, 7, 8, 9, 10),
  y = c(34, 35, 41, 40, 45, 47, 51),
  slope = 2.82,
  p.value = 0.000131
)
```

Extract the second element in the list.

```
second_elm = my.test[2]
second_elm
```

```
## $y
## [1] 34 35 41 40 45 47 51
```

Extract the element named p.value from the list.

```
p_val = my.test["p.value"]
p_val
```

```
## $p.value
## [1] 0.000131
```

(10)Turned in by the due date/time – 2 points.

<3 thank you!