MTH 3270 Exercises I

Brady Lamson

1/22/2022

3: Getting Started with R

3.1: Arithmetic Operators

- Below is a list of mathematical operators ordered from highest to lowest precedence:

Operator	Function
^	Exponentiation
-	Unary minus sign
%%	Modulo
%/%	Integer Division
* /	Multiplication, Division
+ -	Addition, Subtraction

Exercise 1:

Guess what the result of each of the following will be, then check your answers.

a) $4 + 2 * 8$

- I would guess **20**. $2 * 8 = 16$ and $16 + 4 = 20$

b) $4 + 2 * 8 + 3$

- I would guess **23**. $8 * 2$ would go first, and then 3 and 4 would be added afterward.

c) -2^2

- This one trips me up on TI calculators all the time. The exponential goes first, so we'll get a **-4** here. **Always** use paranthesis with exponentials.

d) $1 + 2^2 * 4$

- This is **17**. 2^2 goes first, multiply that by 4 and then add 1.

e) $(2 + 4)/3/2$

- This is **1**. The division by 3 wants to go first, but it can't. The numbers in the paranthesis have to be resolved so division can happen. We get $6/3 = 2$ and $2/2 = 1$.

```
# Tests
result_3.1 <- c(
    4 + 2 * 8,
    4 + 2 * 8 + 3,
    -2^2,
    1 + 2^2 * 4,
    (2 + 4) / 3 / 2
    )

data.table::data.table(
    "exercise" = LETTERS[1:5],
    "result" = result_3.1
    ) %>%
    glue::glue_data("The output of exercise {exercise} is {result}")
```

```
## The output of exercise A is 20
## The output of exercise B is 23
## The output of exercise C is -4
## The output of exercise D is 17
## The output of exercise E is 1
```

3.2: Special Characters, Special Values, Etc.

- R ignores white space.
  - This means you can carry out commands across multiple lines! This is very useful for readability.
- Sometimes you'll have to deal with special values. Two of these are:
  - Inf (Infinity)
  - Nan (Not a number)
- Any positive number divided by 0 will result in **Inf**. 0 divded by 0 results in **NaN**.

Exercise 2:

Guess what the result of each of the following will be, then check your answers.

a) $5/0$

- My guess would be **Inf** based on the note above.

b) $1/\text{Inf}$

- My guess would be **0**. It makes sense to my brain that 1 divded by an arbitrarily large number would approach 0, so we'll go with that!

c) $0/0$

- My guess would be **NaN** based on the note above.

d) $\text{Inf} + 1$

- This will just be **Inf**.

```
# Tests
result_3.2 <- c(
    5 / 0,
    1 / Inf,
    0 / 0,
    Inf + 1
)

data.table::data.table(
    "exercise" = LETTERS[1:4],
    "result" = result_3.2
    ) %>%
    glue::glue_data("The output of exercise {exercise} is {result}")
```

```
## The output of exercise A is Inf
## The output of exercise B is 0
## The output of exercise C is NaN
## The output of exercise D is Inf
```

3.3: Variables and the Assignment Operator

- Atomic Values
  - Numeric
    - Double (Double-precision floating-point, can store integer and non-integer decimals)
    - Integer
  - Character
  - Logical
  - Complex
  - Raw

Another type of variable is NULL, this represents an "empty" variable.

Exercise 3:

What type of variable is created in each of the following commands? Check your answers by typing typeof(x):

a) x <- 45
   - This is an integer.

b) x <- "foo"
   - This is a character

c) x <- FALSE
   - This is a logical

d) x <- NULL
   - This is a NULL

```
# Tests

x <- 45
typeof(x)
```

```
## [1] "double"
```

```
x <- "foo"
typeof(x)
```

```
## [1] "character"
```

```
x <- FALSE
typeof(x)
```

```
## [1] "logical"
```

```
x <- NULL
typeof(x)
```

```
## [1] "NULL"
```

Exercise 4:

Guess the final value of x in the following sequence of commands. Then check your answer.

```
x <- 2
x <- x * 2 + 1
x <- x * 3
# My prediction is that x will be 15.
x
```

```
## [1] 15
```

Exercise 5:

Write commands that do the following (in order):

```
# 1: Create a variable y containing the value 5.
y <- 5

# 2: Overwrite the value of y by the value 3 * y
y <- 3 * y

# 3: Copy the value y into a new variable z
z <- y
```

3.4: Introduction to Functions

Exercise 6:

Look at the help page for sqrt() by typing:

? sqrt

Besides sqrt(), what other R function is described on the help page? - abs() and sqrt() are both described.

Exercise 7:

Look at the help page for signif() by typing:

? signif

From the help page, how many arguments does signif() have? - It has two arguments, **x** and **digits**.

Exercise 8:

Look at the arguments for signif(). This function prints the value passed for x to the number of significant digits specified by digits. a) From the help page, what is the default value for the digits argument? - 6 b) To how many significant digits will the value 342.88937 be printed using the default value for the digits argument? - It will be to 6 significant digits as a digits value is not specified. That would probably be **342.889**.

Exercise 9:

a) Write a command using *named argument matching* that prints the value 342.88937 to 5 significant digits.
b) Write a command using *positional matching* that prints the value 342.88937 to 5 significant digits.

```
signif(digits = 5, x = 342.88937)
```

```
## [1] 342.89
```

```
signif(342.88937, 5)
```

```
## [1] 342.89
```

3.5: The R Workspace

Exercise 10:

Create a few variables named x, y and z. Then type the following sequence of commands, paying attention to the output from ls() each time:

```
x = 1
y = 'a'
z = NaN

ls()
```

```
## [1] "result_3.1" "result_3.2" "x"          "y"          "z"
```

```
rm(x)
ls()
```

```
## [1] "result_3.1" "result_3.2" "y"          "z"
```

```
rm(list = ls())
ls()
```

```
## character(0)
```

The three calls to ls() return everything in the current workspace. So, in that case, it's the pipe I took from magrittr, my previous result lists and the 3 different variables I had just made. The rm() removes the x so that isn't present in the next ls() and then the next rm() removes everything in the current workspace. Because of that, the final ls() simply returns character(0) as nothing is in the workspace. It is an empty list.

3.6: A Preview of R Data Structures

- **Vectors** can store any of the *atomic* types
- **Matrices** are like two-dimensional vectors.
- One limitation of vectors is that everything in them must be the same type. Another type of container, called a **list** does not have that limitation.
  - Lists can contain anything. For example they can contain vectors and even other lists. They're very versatile.
- **Data Frames** are like matrices but can have a mix of *categorical* and *quantitative* data types.
  - It is of note that each *column* of a data frame is a *vector*.
- An **array** is like a matrix but it can have more than two dimensions (e.g. rows, columns, and layers).

Exercise 11:

Write a command using c() that creates a vector containing the values: (3,7,2,8)

```
c(3, 7, 2, 8)
```

```
## [1] 3 7 2 8
```

Exercise 12:

Write a command using matrix() that creates the following matrix:

```
matrix(
    data = seq(from = 2, to = 8, by = 2), nrow = 2, ncol = 2
    )
```

```
##      [,1] [,2]
## [1,]    2    6
## [2,]    4    8
```

Exercise 13:

Write a command using list() that creates a list containing the following elements: ("e", 9, TRUE)

```
list("e", 9, TRUE)
```

```
## [[1]]
## [1] "e"
##
## [[2]]
## [1] 9
##
## [[3]]
## [1] TRUE
```

Exercise 14:

Write a command using data.frame that creates a data frame containing the following data set:

Category	Value
A	5
A	4
B	6
B	6
C	9
C	8

```
data.frame(
    "Category" = sort(rep(LETTERS[1:3], 2)),
    "Value" = c(5, 4, 6, 6, 9, 8)
)
```

```
##   Category Value
## 1        A     5
## 2        A     4
## 3        B     6
## 4        B     6
## 5        C     9
## 6        C     8
```

4: Vectors

4.2: Vector Arithmetric and Recycling

Exercise 15:

Guess the result of the following code.

This code snippet will simply combine the two vectors resulting in a single vector of 2:9.

```
x <- c(2, 3, 4, 5)
y <- c(6, 7, 8, 9)
c(x, y)
```

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

This code snippet will result in a vector of (8, 10, 12, 14).

```
x + y
```

```
## [1]  8 10 12 14
```

Exercise 16:

Guess the result of the following code.

This code snippet will add 1 to the vector x, giving (3, 4, 5, 6)

```
x <- c(2, 3, 4, 5)
x + 1
```

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

This code snippet will multiply all values in the vector by 2, giving (4, 6, 8, 10)

```
x * 2
```

```
## [1]  4  6  8 10
```

Exercise 17:

Guess the result of the following code.

This code snippet will utilize some recycling. You'll get $6+2$, $7+3$ and, due to recycling, $8+2$. As a result we'll get (8, 10, 10)

```
y <- c(6, 7, 8)
z <- c(2, 3)
y + z
```

```
## Warning in y + z: longer object length is not a multiple of shorter object
## length
```

```
## [1]  8 10 10
```

Exericse 18:

Guess the result of the following code.

Both is.vector() functions will return TRUE. That's due to single-valued variables and constants being vectors of length 1, and also the fact that both *a* and *b* here are equivalent operations.

```
# a
x <- 2
is.vector(x)
```

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

```
# b
is.vector(2)
```

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

4.3: Vector Coercion

- All elements of a vector must be the same type, so if you try to combine vectors of different types, they'll be ***coerced*** to the most *flexible* type. Types from least to most flexible are:

Flexibility	Type
1 (Least Flexible)	Logical
2	Integer
3	Double
4 (Most Flexible)	Character

Exercise 19:

Guess the result of the following code.

a) The below code will result in a vector of ("2", "3", "a") due to integers being less flexible than characters.

```
x <- c(2, 3, "a")
x
```

```
## [1] "2" "3" "a"
```

b) Logicals are the least flexible, so TRUE will be coerced into a 1. The result will be (2, 3, 1)

```
x <- c(2, 3, TRUE)
x
```

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

c) Logicals are the least flexible so they'll be coerced to strings. The result will be ("a", "b", "FALSE", "TRUE")

```
x <- c("a", "b")
y <- c(FALSE, TRUE)
c(x, y)
```

```
## [1] "a"     "b"     "FALSE" "TRUE"
```

4.4: Common Vector Operations

Exercise 20:

Consider the following vector:

```
x <- c(7, 6, 4, 2, 3, 5)
```

Guess the result of the following code.

a) x[2]
   - This will be the second index, so **6**.

b) x[-2]
   - This will be all *but* the second index, so **(7, 4, 2, 3, 5)**

c) x[c(1, 2)]
   - This is the first and second index, so **(7, 6)**

d) x[c(2, 1)]
   - This is the second and first index, so **(6, 7)**

e) x[1] <- 5
   - This changes the first index, so x is now **(5, 6, 4, 2, 3, 5)**

```
x[2]
```

```
## [1] 6
```

```
x[-2]
```

```
## [1] 7 4 2 3 5
```

```
x[c(1,2)]
```

```
## [1] 7 6
```

```
x[c(2,1)]
```

```
## [1] 6 7
```

```
x[1] <- 5
x
```

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

Exercise 21:

Consider the following vector:

```r
x <- c(7, 6, 4, 2, 3, 5)

# a) Write a command that returns the 4th element of x
x[4]
```

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

```r
# b) Write a command that replaces the 4th element of x with the value 1
x[4] <- 1

# c) Write a command that returns all but the 6th element of x
x[-6]
```

```
## [1] 7 6 4 1 3
```

Exercise 22:

Guess the result of the following code:

```r
x <- c(7, 6, 4, 2)
ans <- x[c(2, 1, 3, 4)]
```

This code snippet is simply reordering the vector. What we get from this is **(6, 7, 4, 2)**

```r
ans
```

```
## [1] 6 7 4 2
```

Exercise 23:

Consider the following vector:

```r
x <- c(7, 6, 4, 2, 3, 5)

# a) Sort the vector in ascending order.
sort(x)
```

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

```r
# b) Sort the vector in reverse order.
rev(x)
```

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

```r
# c) Sort the vector in descending order.
sort(x, decreasing = TRUE)
```

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

Exercise 24:

Consider the same vector as previous. Guess the result of the following code.

```
ans <- x[c(FALSE, FALSE, TRUE, FALSE, FALSE, TRUE)]
```

This code snippet will return only the values with TRUE at their index. So we'll get **(4, 5)**.

```
ans
```

```
## [1] 4 5
```

Exercise 25:

Guess the result of the following code:

a) 1:5
   - This will return a vector 1 through 5.

b) 6:10
   - This will return a vector 6 through 10.

c) 5:1
   - This will return a vector 5 through 1, descending.

```
1:5
```

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

```
6:10
```

```
## [1]  6  7  8  9 10
```

```
5:1
```

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

Exercise 26:

Guess the result of the following code:

is.vector(1:5) - My guess would be this returning TRUE.

```
is.vector(1:5)
```

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

Exercise 27:

Guess the result of the following code:

a) seq(from = 1, to = 2.5, by = 0.5)
   - (1, 1.5, 2, 2.5)

b) seq(from = 2.5, to = 1, by = -0.5)
   - (2.5, 2, 1.5, 1)

```
seq(from = 1, to = 2.5, by = 0.5)
```

```
## [1] 1.0 1.5 2.0 2.5
```

```
seq(from = 2.5, to = 1, by = -0.5)
```

```
## [1] 2.5 2.0 1.5 1.0
```

Exercise 28:

Guess the result of the following code:

a) rep(2, times = 3)
   - (2, 2, 2)

b) rep(1:2, times = 3)
   - (1, 2, 1, 2, 1, 2)

```
rep(2, times = 3)
```

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

```
rep(1:2, times = 3)
```

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

4.5: Comparison Operators

Exercise 29:

Consider the following vector:

```
x <- c(3, 4, 10)
```

Guess the result of the following code. a) x == 4 - (FALSE, TRUE, FALSE) b) x > 4 - (FALSE, FALSE, TRUE) c) x >= 4 - (FALSE, TRUE, TRUE) d) x != 4 - (TRUE, FALSE, TRUE)

```
x == 4
```

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

```
x > 4
```

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

```
x >= 4
```

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

```
x != 4
```

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

Exercise 30:

Consider the following two vectors:

```
x <- c(3, 4, 10)
y <- c(3, 4, 5)
```

Guess the result of the following code: a) x == y - (TRUE, TRUE, FALSE) b) x != y - (FALSE, FALSE, TRUE)

```
x == y
```

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

```
x != y
```

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

Exercise 31:

Guess the result of the following code: a) TRUE + TRUE + FALSE + FALSE + FALSE - 2 b) sum(c(TRUE, TRUE, FALSE, FALSE, FALSE)) - 2

```
TRUE + TRUE + FALSE + FALSE + FALSE
```

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

```
sum(c(TRUE, TRUE, FALSE, FALSE, FALSE))
```

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

Exercise 32:

Consider the following vector:

```
x <- c(10, 8, -2, -6, -5)
```

Guess the result of the following code: a) x > 0 - (TRUE, TRUE, FALSE, FALSE, FALSE) b) sum(x > 0) - 2

```
x > 0
```

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

```
sum(x > 0)
```

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

4.6: Using any(), all(), and which(), which.min(), and which.max()

Exercise 33:

Consider the vector

```
x <- c(2,8,6,7,1,4,9)
```

Guess the result of the following code: a) any(x == 4) - TRUE b) all(x == 4) - FALSE c) which(x == 4) - 6 d) which(x != 4) - (1, 2, 3, 4, 5, 7)

```
# Tests
any(x == 4)
```

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

```
all(x == 4)
```

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

```
which(x == 4)
```

```
## [1] 6
```

```
which(x != 4)
```

```
## [1] 1 2 3 4 5 7
```

Exercise 34:

Consider the following vectors:

```
x <- c(53, 42, 64, 71, 84, 62, 95)
y <- c(53, 41, 68, 71, 81, 66, 65)
```

a) Write a command involving any() and == to determine if *any* of the values in x are equal to their corresponding value in y.

```
any(x == y)
```

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

b) Write a command involving all() and == to determine if *all* of the values in x are equal to their corresponding value in y.

```
all(x == y)
```

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

c) Write a command involving which() and == to determine *which* of the values in x are equal to their corresponding value in y.

```
which(x == y)
```

```
## [1] 1 4
```

Exercise 35:

Consider the x vector from the previous problem. Guess the output of the following code:

a) which.min(x)
   - 42

b) which.max(x)
   - 95

```
which.min(x)
```

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

```
which.max(x)
```

```
## [1] 7
```

4.7: Computing Summary Statistics

Exercise 36:

Consider the following data set:

```
x <- c(10, 147, 7, 6, 7, 12, 9, 12, 11, 8)

# a) Use mean() to compute the mean
mean(x)
```

```
## [1] 22.9
```

```
# b) Use median() to calculate the median
median(x)
```

```
## [1] 9.5
```

```
# c) Use sd() to calculate the standard deviation
sd(x)
```

```
## [1] 43.65636
```

Exercise 37:

The standard deviation measures variation in a set of data.

a) What do you think the standard deviation of the following dataset will be? u <- c(5, 5, 5, 5, 5)

- 0. All the values are the same so there is no variation.

```
sd(c(5, 5, 5, 5, 5))
```

```
## [1] 0
```

b) Which of the following two data sets do think will have a higher standard deviation?

- The second one will have the higher standard deviation. The means of the two data sets are the same, but the second data set has values far further away from said mean.

```
sd(c(5, 6, 7))
```

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

```
sd(c(1, 6, 11))
```

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

4.8: Vectorized Computations

Exercise 38:

The function abs() takes the absolute value of a number. Guess the result of the following command. - (1, 3, 4, 2)

```
abs(c(-1, 3, -4, -2))
```

```
## [1] 1 3 4 2
```

Exercise 39:

Consider the following temperature measurements, in degree Celsius:

```
degreesC <- c(23, 19, 21, 22, 18, 20, 24, 25)
```

The relationship between Celsius ($^\circ C$) and Fahrenheit ($^\circ F$) is:

$$^\circ F = \frac{9}{5} \cdot^\circ C + 32$$

Recall that arithmetic operators such as * and + are *vectorized*. Describe in words what the following command will do to the Celsius temperatures. Try it. - The following code will multiply each individual value by $\frac{9}{5}$ and then add 32 to them, thus converting each value in the vector from Celsius to Fahrenheit and assigning that to a new variable, degreesF.

```
degreesF <- (9/5) * degreesC + 32
degreesF
```

```
## [1] 73.4 66.2 69.8 71.6 64.4 68.0 75.2 77.0
```

4.9: Filtering

Exercise 40:

Consider again the vector:

```
x <- c(538, 432, 684, 716, 814, 624, 956)
```

Guess the result of the following code: a) x[x > 700] - (716, 814, 956) b) subset(x, subset = x > 700) - (716, 814, 956) c) Write a command involving square brackets [ ] that extracts from x all values that are *not equal* to 814

```
x[x > 700]
```

```
## [1] 716 814 956
```

```
subset(x, subset = x > 700)
```

```
## [1] 716 814 956
```

```
# c)
subset(x, subset = x != 814)
```

```
## [1] 538 432 684 716 624 956
```

Exercise 41:

Consider this data set:

Gender	Age	Blood Pressure
f	33	118
m	35	115
f	29	110
m	34	117
m	37	112
f	36	119
f	35	114
f	40	121
m	43	123
f	38	117
f	40	120
m	44	121

```
Gender <- c("f", "m", "f", "m", "m", "f", "f", "f", "m", "f", "f", "m")
Age <- c(33, 35, 29, 34, 37, 36, 35, 40, 43, 38, 40, 44)
BP <- c(118, 115, 110, 117, 112, 119, 114, 121, 123, 117, 120, 121)

ex_41_df <- data.table::data.table(
    "Gender" = Gender,
    "Age" = Age,
    "Blood Pressure" = BP
) %>% tibble::as_tibble()

ex_41_df
```

```
## # A tibble: 12 x 3
##    Gender   Age `Blood Pressure`
##    <chr>  <dbl>            <dbl>
##  1 f         33              118
##  2 m         35              115
##  3 f         29              110
##  4 m         34              117
##  5 m         37              112
##  6 f         36              119
##  7 f         35              114
##  8 f         40              121
##  9 m         43              123
## 10 f         38              117
## 11 f         40              120
## 12 m         44              121
```

One of the commands below extracts the blood pressure of just the males, the other extracts the ages of those whose blood pressure exceeds 117. Which one is which?

a) Age[BP $>$ 117]

- This one extracts the ages of people with a blood pressure of more than 117.

b) BP[Gender == “m”]

- This one extracts the blood pressure of the males in the data set.

4.10: NA Values

Exercise 42:

Guess the result of the following code: a) 3 == NA - NA b) NA == NA - NA

```
3 == NA
```

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

```
NA == NA
```

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

Exercise 43:

Consider the vector:

```
x <- c(1, 2, NA)
```

Guess the result of the following code: a) is.na(x) - FALSE FALSE TRUE b) x[is.na(x)] <- 0 - (1, 2, 0)

Exercise 44:

Consider the same vector as in Exercise 43:

Guess the result of the following code: a) sum(x) - NA b) sum(x, na.rm = TRUE) - 3

```
sum(x)
```

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

```
sum(x, na.rm = TRUE)
```

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

5: Matrices

5.1: Creating and Examining Matrices

Exercise 45:

Here's a matrix x

```
x <- c(8, 4) %>%
    rep(3) %>%
    sort(decreasing = TRUE) %>%
    matrix(nrow = 2, byrow = TRUE)
x
```

```
##      [,1] [,2] [,3]
## [1,]    8    8    8
## [2,]    4    4    4
```

Guess the result of the following code: a) dim(x) - 2, 3 b) nrow(x) - 2 c) ncol(x) - 3

```
dim(x)
```

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

```
nrow(x)
```

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

```
ncol(x)
```

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

Exercise 46:

What happens when you run the following command? - matrix(c(1,2,3), nrow = 4, ncol = 2)

My guess:

$$\begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 1 \\ 1 & 2 \end{bmatrix}$$

```
matrix(c(1,2,3), nrow = 4, ncol = 2)
```

```
## Warning in matrix(c(1, 2, 3), nrow = 4, ncol = 2): data length [3] is not a sub-
## multiple or multiple of the number of rows [4]
```

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

Exercise 47:

Consider the matrix:

$$\begin{bmatrix} 5 & 5 \\ 4 & 4 \end{bmatrix}$$

Do you think the following commands will produce the matrix above? a) x <- matrix(c(5, 4, 5, 4), nrow = 2, ncol = 2) - Yes I do. Since it goes by column this should work just fine. b) x <- cbind(c(5, 4), c(5, 4)) - This will work as well. c) x <- rbind(c(5, 5), c(4, 4)) - I think even this one works

```
matrix(c(5, 4, 5, 4), nrow = 2, ncol = 2)
```

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

```
cbind(c(5, 4), c(5, 4))
```

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

```
rbind(c(5, 5), c(4, 4))
```

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

5.2 General Matrix Operations

Exercise 48: Consider the following matrix

```
x <- matrix(1:9, nrow = 3, ncol = 3)
x
```

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

Guess the result of the following code: a) x[1, 3] - 7

b) x[1,]

- 1, 4, 7

c) x[, 3]

- 7, 8, 9

d) x[, -3]

- My guess: $\begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$

```
# Tests

x[1, 3]
```

```
## [1] 7
```

```
x[1,]
```

```
## [1] 1 4 7
```

```
x[, 3]
```

```
## [1] 7 8 9
```

```
x[, -3]
```

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

Exercise 49:

Consider the following matrix:

```
x <- matrix(1:6, nrow = 2, ncol = 3)
x
```

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

What does the following command do? x[, c(3,1,2)] - This will return the 3rd column, the 1st column, followed by the 2nd column. It may look like: $\begin{pmatrix} 5 & 1 & 3 \\ 6 & 2 & 4 \end{pmatrix}$

```
x[, c(3,1,2)]
```

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

5.3: The apply() Function

Exercise 50:

Consider the following matrix x:

```
x <- matrix(
    c(8, 6, 3, 6, 5, 7),
    nrow = 3, ncol = 2
    )
x
```

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

Guess the result of the following code: a) apply(x, MARGIN = 1, FUN = sum) - MARGIN 1 indicates the function will operate over the rows. Therefore, I would think we would get: - 14, 11, 10 b) apply(x, MARGIN = 2, FUN = min) - MARGIN 2 indicates we're operating over columns this time. So we'll get the minimum of each column. - 3, 5

```
apply(x, MARGIN = 1, FUN = sum)
```

```
## [1] 14 11 10
```

```
apply(x, MARGIN = 2, FUN = min)
```

```
## [1] 3 5
```

Exercise 51:

a) Using the *USPersonalExpenditure* data set, which command will find the mean expenditure for each of the five expenditure categories?

- apply(X = USPersonalExpenditure, MARGIN = 1, FUN = mean)
- apply(X = USPersonalExpenditure, MARGIN = 2, FUN = mean)
  - The **first** one would give what we want. MARGIN = 1 has the function work over the rows.

```
apply(X = USPersonalExpenditure, MARGIN = 1, FUN = mean)
```

```
##    Food and Tobacco Household Operation  Medical and Health       Personal Care
##              57.260              27.540              10.820              2.854
##   Private Education
##               1.871
```

```
apply(X = USPersonalExpenditure, MARGIN = 2, FUN = mean)
```

```
##    1940    1945    1950    1955    1960
##  7.5222 13.7428 20.5120 25.9400 32.6280
```

b) Using the *USPersonalExpenditure* data set, which command will find the mean expenditure for each of the five expenditure categories?

- apply(X = USPersonalExpenditure, MARGIN = 1, FUN = sum)
- apply(X = USPersonalExpenditure, MARGIN = 2, FUN = sum)
  - The second is what we want, MARGIN = 2 operates over columns, which will give us information on the years.

```
apply(X = USPersonalExpenditure, MARGIN = 1, FUN = sum)
```

```
##    Food and Tobacco Household Operation  Medical and Health       Personal Care
##             286.300             137.700              54.100              14.270
##   Private Education
##               9.355
```

```
apply(X = USPersonalExpenditure, MARGIN = 2, FUN = sum)
```

```
##    1940    1945    1950    1955    1960
##  37.611  68.714 102.560 129.700 163.140
```

6: Lists

6.1: Creating and Examining Lists

Exercise 52

Using the following list: a) Use str() to look at the structure of the list. Report the results. - Employees is made up of 3 things, a character vector of names, a number vector of salaries and a vector of logicals. b) Use length() to find the number of elements in the list. Report the results. - There are 3 elements in the list. This makes sense as we have names, salaries and union. Those vectors individual lengths is not accounted for here.

```
Employees <- list(Name = c("Joe", "Kim", "Ann", "Bob"),
Salary = c(56000, 67000, 60000, 55000),
Union = c (TRUE, TRUE, FALSE, FALSE))

str(Employees)
```

```
## List of 3
##  $ Name  : chr [1:4] "Joe" "Kim" "Ann" "Bob"
##  $ Salary: num [1:4] 56000 67000 60000 55000
##  $ Union : logi [1:4] TRUE TRUE FALSE FALSE
```

```
paste("---")
```

```
## [1] "---"
```

```
length(Employees)
```

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

6.2: General List Operations

Exercise 53:

Using the Employees list, guess the results of the following code: a) Employees[[2]] - This returns the second item in this list, in this case it'd be the number vector for Salary. b) Employees$Salary - This is equivalent to problem a). c) Write a command involving [[ ]] that returns the "logical" vector Union from the Employees list d) Now write a command involving $ that returns the "logical" vector Union from the Employees list.

```
# Tests
## c)
Employees[[3]]
```

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

```
print("---")
```

```
## [1] "---"
```

```
## d)
Employees$Union
```

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

6.3: Named List Elements

Exercise 54:

Here's a simple list x:

```
x <- list(x1 = c(1, 2), x2 = c("a", "b"), x3 = 12)
```

What will the following command return? Check your answer. - This will return the names of the list, so x1, x2 and x3.

```
names(x)
```

```
## [1] "x1" "x2" "x3"
```

6.4: Applying a function to a list using lapply() and sapply().

Exercise 55:

Heres the HtWtAge list again

```
HtWtAge <- list(
    Height = c(65, 68, 70, 60, 61),
    Weight = c(160, 171, 158, 148, 215),
    Age = c(23, 20, 37, 40, 44)
)
```

a) Write a command using lapply(), with FUN = max, that returns a *list* containing the maximum value of each variable (Height, Weight, and Age).

b) Now write a command using sapply() that does the same thing, but returns a *vector*.

```
# a)
lapply(HtWtAge, FUN = max)
```

```
## $Height
## [1] 70
##
## $Weight
## [1] 215
##
## $Age
## [1] 44
```

```
# b)
sapply(HtWtAge, FUN = max)
```

```
## Height Weight    Age
##     70    215     44
```

7: Data Frames

7.1: Creating and Viewing Data Frames

Exercise 56:

a) Here's the **mice data set** as three vectors:

```
col <- c("white", "grey", "black", "brown", "black", "white", "black",
"white")
wt <- c(23, 21, 12, 26, 25, 22, 26, 19)
len <- c(3.8, 3.7, 3.0, 3.4, 3.4, 3.1, 3.5, 3.2)
```

After creating the vectors, write a command involving data.frame() that creates a data frame containing the data. Make sure the column names are Color, Weight and Length.

```
mice.data <- data.frame(
    "Color" = col,
    "Weight" = wt,
    "Length" = len
)

mice.data
```

```
##   Color Weight Length
## 1 white     23    3.8
## 2  grey     21    3.7
## 3 black     12    3.0
## 4 brown     26    3.4
## 5 black     25    3.4
## 6 white     22    3.1
## 7 black     26    3.5
## 8 white     19    3.2
```

b) Before proceeding, remove the data frame you just created from your workspace.

```
rm(mice.data)
```

the file **mice.txt** contains the mice data set. Load the code into R and show the data set.

```
#my.file <- file.choose()
```

```
# Breaks when knitting to pdf.
#my.file
```

c) Save that file to a csv and run the csv.

```
# This breaks on
#mice.data <- read.csv(my.file, sep = "", header = TRUE)
```

```
#mice.data
```

Note: Due to the nature of file.choose() this breaks when the work environment is altered. An actual workable solution will be used instead.

```
mice.data <- read.delim("mice.txt", sep = "") %>%
    as.data.frame()
```

d) Now type the following commands and report the results:

```
nrow(mice.data)
```

```
## [1] 8
```

```
ncol(mice.data)
```

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

```
head(mice.data)
```

```
##   Color Weight Length
## 1 white     23    3.8
## 2  grey     21    3.7
## 3 black     18    3.0
## 4 brown     26    3.4
## 5 black     25    3.4
## 6 white     22    3.1
```

```
names(mice.data)
```

```
## [1] "Color"  "Weight" "Length"
```

```
str(mice.data)
```

```
## 'data.frame':    8 obs. of  3 variables:
##  $ Color : chr  "white" "grey" "black" "brown" ...
##  $ Weight: int  23 21 18 26 25 22 26 19
##  $ Length: num  3.8 3.7 3 3.4 3.4 3.1 3.5 3.2
```

7.2: Accessing and Replacing Elements, Rows, or Columns of a Data Frame

Exercise 57: Consider the following data on nine people. The following commands will create a data frame containing the data.

```
status <- c("Married", "Single", "Single", "Married", "Single",
"Married", "Married", "Single", "Single")
age <- c(36, 33, 21, 29, 19, 35, 39, 28, 21)
educ <- c("HS Diploma", "Bachelor of Arts", "Bachelor of Science",
"Bachelor of Science", "HS Diploma", "Bachelor of Arts",
"Master of Science", "HS Diploma", "HS Diploma")

my.data <- data.frame(Status = status, Age = age, Education = educ)
```

a) Guess what each of the following commands will return, then check your answers.

- my.data[6,2]
  - This will return the information in row 6 column 2, so the age **35** I think.
- my.data[6,]
  - This will return everything in row 6. So thats **Married, 35, Bachelor of Arts**
- my.data[, 2]
  - This will return everything in column 2, so the entire set of Ages.
- my.data$Age
  - This is equivalant to the above code. It will return the entire set of ages.
- my.data[[2]]
  - I *think* this will return the entire second row? So **Single, 33, Bachelor of Arts**.

```
my.data[6, 2]
```

```
## [1] 35
```

```
my.data[6, ]
```

```
##    Status Age        Education
## 6 Married  35 Bachelor of Arts
```

```
my.data[, 2]
```

```
## [1] 36 33 21 29 19 35 39 28 21
```

```
my.data$Age
```

```
## [1] 36 33 21 29 19 35 39 28 21
```

```
my.data[[2]]
```

```
## [1] 36 33 21 29 19 35 39 28 21
```

b) Write three different commands that return the entire 3rd column (Education) of the data frame.

```
# First solution []
my.data[, 3]
```

```
## [1] "HS Diploma"          "Bachelor of Arts"    "Bachelor of Science"
## [4] "Bachelor of Science" "HS Diploma"          "Bachelor of Arts"
## [7] "Master of Science"   "HS Diploma"          "HS Diploma"
```

```
print("---[Second Solution]---")
```

```
## [1] "---[Second Solution]---"
```

```
# Second solution $
my.data$Education
```

```
## [1] "HS Diploma"          "Bachelor of Arts"    "Bachelor of Science"
## [4] "Bachelor of Science" "HS Diploma"          "Bachelor of Arts"
## [7] "Master of Science"   "HS Diploma"          "HS Diploma"
```

```
print("---[Third Solution]---")
```

```
## [1] "---[Third Solution]---"
```

```
# Third solution [[]]
my.data[[3]]
```

```
## [1] "HS Diploma"          "Bachelor of Arts"    "Bachelor of Science"
## [4] "Bachelor of Science" "HS Diploma"          "Bachelor of Arts"
## [7] "Master of Science"   "HS Diploma"          "HS Diploma"
```

c) The nine people have each aged one year since the data were collected. Here's a vector containing their current ages:

```
age2 <- c(37, 34, 22, 30, 20, 36, 40, 29, 22)
```

Describe in words what the following commands do

- my.data$Age <- NULL

    - Replaces all ages with NULL

- my.data$Age2 <- age2

    - Creates a new column, Age2 with the values in the age2 vector.

d) Here's another vector:

ageofspouse <- c(39, NA, NA, 34, NA, 27, 30, NA, NA)

```
- my.data$AgeOfSpouse <- ageofspouse

    - Creates a new column, AgeOfSpouse with the values in the ageofspouse vector

- my.data[["AgeOfSpouse"]] <- ageofspouse

    - Equivalant to the previous set of code.
```

7.3: Viewing and Changing Variable Names in a Data Frame

Exercise 58:

Given the data frame:

```
x <- data.frame(A = 1:5, B = 6:10, C = c("a", "b", "c", "d", "e"))
```

a) Guess what each of the following commands will return, then check your answers.

- names(x)
  - The names of the data.frame, so **A, B and C**.
- is.vector(names(x))
  - It should be **TRUE**
- typeof(names(x))
  - **Character**

```
names(x)
```

```
## [1] "A" "B" "C"
```

```
is.vector(names(x))
```

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

```
typeof(names(x))
```

```
## [1] "character"
```

b) Guess what the following will do, then check your answer.

- names(x) <- c(“AA”, “BB”, “CC”)
  - Renames the columns to those character strings.

```
names(x) <- c("AA", "BB", "CC")
x
```

```
##   AA BB CC
## 1  1  6  a
## 2  2  7  b
## 3  3  8  c
## 4  4  9  d
## 5  5 10  e
```

7.5: Filtering on Data Frames

Exercise 59:

a) Write a command involving square brackets [ ] that returns just the rows of **warpbreaks** corresponding to observations made at the "M" level of tension.

```
# Note: Head used to avoid bloating pdf even further.
head(warpbreaks[warpbreaks$tension %in% "M", ])
```

```
##    breaks wool tension
## 10     18    A       M
## 11     21    A       M
## 12     29    A       M
## 13     17    A       M
## 14     12    A       M
## 15     18    A       M
```

b) Now write a command that uses subset() to do the same thing as in part a.

```
head(subset(warpbreaks, subset = tension %in% "M"))
```

```
##    breaks wool tension
## 10     18    A       M
## 11     21    A       M
## 12     29    A       M
## 13     17    A       M
## 14     12    A       M
## 15     18    A       M
```

8: Factors

Creating and Viewing Factors and Their Levels

Exercise 60:

a) Write a command that uses factor() to convert the following "character" vector

```
x.factor <- c("a", "a", "b", "b", "c", "c", "d", "d") %>%
    as.factor()

x.factor
```

```
## [1] a a b b c c d d
## Levels: a b c d
```

b) After creating the factor **x.factor**, guess what the result of the following code will be.

- length(x.factor)
  - I *think* it would give us 8 as the length.

```
length(x.factor)
```

```
## [1] 8
```

c) Guess what the result of the following command will be, then check your answer.

- levels(x.factor)
  - Looking at the output of x.factor it gives us the levels so I would imagine this would return the same thing. **a b c d**

```
levels(x.factor)
```

```
## [1] "a" "b" "c" "d"
```

d) Guess what the result of the following command will be.

- nlevels(x.factor)
  - Based on the factors it gave us, it seems that there are **4** of them.

```
nlevels(x.factor)
```

```
## [1] 4
```

Exercise 61:

Guess the result of the following code:

```
- The code will probably return "a", "b", "c".
```

```r
x.fac <- factor(c("a", "a", "a", "b", "b", "c"))
levels(x.fac)
```

```
## [1] "a" "b" "c"
```

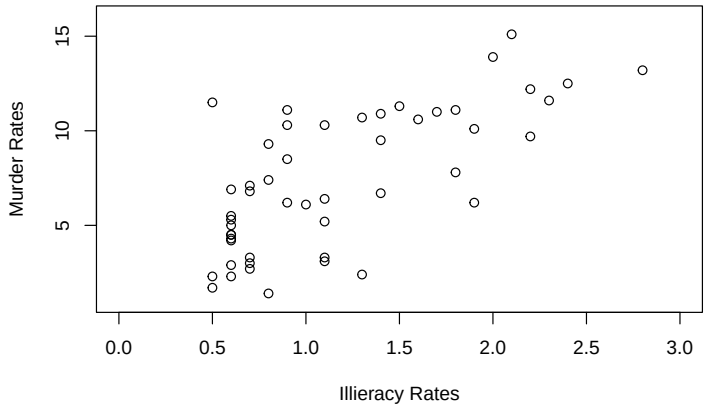
9: Data Visualization With Base R

9.1: Creating Plots

Exercise 62:

Use plot() to make a scatterplot of the murder rates (y-axis) versus illiteracy rates (x-axis) of the state.x77 data set.

```
illit <- state.x77[, 3]
murder <- state.x77[, 5]
plot(x = illit, y = murder,
     ylab = "Murder Rates",
     xlab = "Illieracy Rates",
     xlim = c(0, 3),
     ylim = c(1, 16))
```



Exercise 63:

Given the vector:

```
laughed <- c("Yes", "Yes", "Yes", "No", "Yes", "No", "Yes", "Yes", "Yes", "Yes", "No",
"Yes", "Yes", "Yes", "No", "No", "Yes", "Yes", "Yes", "No", "Yes")
```

a) In words, what do the following commands do:

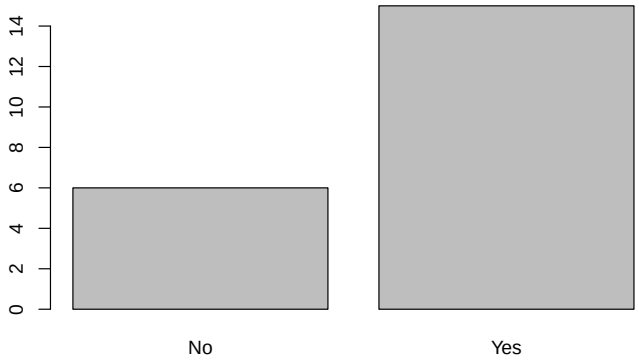
- my.tab <- table(laughed)
- my.tab
  - Creates a table using the vector, creating two columns out of the number of Yes and No counts.

```
my.tab <- table(laughed)
my.tab
```

```
## laughed
##  No Yes
##   6  15
```

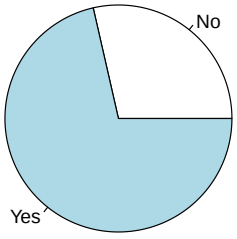
b) Pass the *table* object my.tab to barplot() to make a bar plot of the counts. Report your R command.

```
my.tab %>% barplot()
```



c) Pass the *table* object my.tab to pie() to make a pie chart of the counts.

```
my.tab %>% pie()
```



10: Objects and Classes

Exercise 64:

Guess the class of each of the objects below.

a) class(c("a", "b", "c"))

- I would assume that this is a character vector. It's a vector of characters after all!

b) class(c(3, 6, 1))

- This would be a numeric vector by the same logic.

c) class(list(3, 5, 2))

- I believe this is just a list.

```
c("a", "b", "c") %>%
    class()
```

```
## [1] "character"
```

```
c(3, 6, 1) %>%
    class()
```

```
## [1] "numeric"
```

```
list(3, 5, 2) %>%
    class()
```

```
## [1] "list"
```

Exercise 65:

a) Guess the class of **sqrt()**, then check your answer.

- I would assume this would have the class of *function*.

```
sqrt %>%
    class()
```

```
## [1] "function"
```

b) Guess the result of the command **is.function(sqrt)**.

- This should return TRUE

```
sqrt %>%
    is.function()
```

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

c) Using the following function, **apply.fun()**, guess the result of the following commands.

- apply.fun(x = 4, FUN = sqrt)
  - This will take the square root of 4 and return 2.
- apply.fun(x = -3, FUN = abs)
  - This will take the absolute value of -3 and return 3.

```
## The output of apply.fun(x = 4, FUN = sqrt) is 2.
## The output of apply.fun(x = -3, FUN = abs) is 3.
```

Exercise 66

Using the below variables, guess the result of the following commands.

```
x <- c("a", "b", "c")
y <- factor(c("a", "b", "c"))
```

a)
- class(x)
  - Based on previous exercises we know this returns "Character"
- typeof(x)
  - Based on the available types I feel like this would *also* return character.

b)
- class(y)
  - This should return "factor"
- typeof(y)
  - I think this would return "character"

```
## x has the class character and is of type character
## y has the class factor and is of type integer
```

11: Generic Functions and Methods

11.2: Methods

Exercise 67:

a) Use methods() to look at **summary()**'s methods. Do the same for **data.frame** and **factor**.

```
summary %>%
     methods()
```

```
## Warning in .S3methods(generic.function, class, envir): generic function '.'
## dispatches methods for generic 'summary'
```

```
##  [1] summary.aov                          summary.aovlist*
##  [3] summary.aspell*                      summary.check_packages_in_dir*
##  [5] summary.connection                   summary.data.frame
##  [7] summary.Date                         summary.default
##  [9] summary.ecdf*                        summary.factor
## [11] summary.glm                          summary.infl*
## [13] summary.lm                           summary.loess*
## [15] summary.manova                       summary.matrix
## [17] summary.mlm*                         summary.nls*
## [19] summary.packageStatus*               summary.POSIXct
## [21] summary.POSIXlt                      summary.ppr*
## [23] summary.prcomp*                      summary.princomp*
## [25] summary.proc_time                    summary.rlang_error*
## [27] summary.rlang_message*               summary.rlang_trace*
## [29] summary.rlang_warning*               summary.rlang:::list_of_conditions*
## [31] summary.srcfile                      summary.srcref
## [33] summary.stepfun                      summary.stl*
## [35] summary.table                        summary.tukeysmooth*
## [37] summary.vctrs_sclr*                  summary.vctrs_vctr*
## [39] summary.warnings
## see '?methods' for accessing help and source code
```

```
data.frame %>%
    methods()
```

```
## no methods found
```

```
factor %>%
    methods()
```

```
## no methods found
```

b) Try the following commands and describe the results.

```
x <- data.frame(x1 = c(4, 3, 6),
x2 = c(4, 7, 9),
x3 = c(1, 1, 3))
summary(x)
```

```
##        x1              x2              x3
##  Min.   :3.000   Min.   :4.000   Min.   :1.000
##  1st Qu.:3.500   1st Qu.:5.500   1st Qu.:1.000
##  Median :4.000   Median :7.000   Median :1.000
##  Mean   :4.333   Mean   :6.667   Mean   :1.667
##  3rd Qu.:5.000   3rd Qu.:8.000   3rd Qu.:2.000
##  Max.   :6.000   Max.   :9.000   Max.   :3.000
```

```
y <- factor(c("a", "a", "b", "c", "c", "c", "c"))
summary(y)
```

```
## a b c
## 2 1 4
```

Summary for x works on each column individually. This results in separate summary statistics for x1, x2 and x3. The summary for y works a bit differently. It seems to have just counted up the number of each "a", "b" and "c".

c) Compare the results of the following commands to those in part (b).

```
summary.data.frame(x)
```

```
##        x1              x2              x3
##  Min.   :3.000   Min.   :4.000   Min.   :1.000
##  1st Qu.:3.500   1st Qu.:5.500   1st Qu.:1.000
##  Median :4.000   Median :7.000   Median :1.000
##  Mean   :4.333   Mean   :6.667   Mean   :1.667
##  3rd Qu.:5.000   3rd Qu.:8.000   3rd Qu.:2.000
##  Max.   :6.000   Max.   :9.000   Max.   :3.000
```

```
summary.factor(y)
```

```
## a b c
## 2 1 4
```

The output here is identical to part (b). This makes sense as we're essentially doing the same thing, we just skip the middle man by applying the correct method of summary directly.

13: Some R Programming Features

13.1: The Logical Operations "And", "Or", and "Not"