Problem Set 1 Solutions

Introduction to R | University of Oxford Sociology

Casey Breen

Problem Set 1

This problem set includes all of the in-class group exercises from session 1. Please try completing these exercises independently.

Complete the following questions in R within a Quarto document.

Exercise 1: Assignment, Arithmetic, Logical Expressions

1.1

Assign `x` and `y` to take values 3 and 4.

```
# Assign x and y to take values 3 and 4
x <- 3
y <- 4
```

1.2

Assign `z` as the product of `x` and `y`.

```
# Create a new variable z as the product of variables x and y
z <- x * y
```

1.3

Calculate the square of 3 and assign it to `three_squared`.

```r
# Write code to calculate the square of 3 and assign it to a variable three_squared
three_squared <- 3^2
```

1.4

Write a logical expression to check if `three_squared` is greater than 10.

```r
# Write a logical expression to check if `three_squared` is greater than 10
three_squared > 10
```

```
[1] FALSE
```

1.5

Write a logical expression to test whether `three_squared` is *not* greater than 10. Use the negate (`!`) operator.

```r
# Write a logical expression to check if `three_squared` is not greater than 10
!three_squared > 10
```

```
[1] TRUE
```

Exercise 2: Sequencing

2.1

Generate vectors containing the numbers 100 to 105 using three different methods (`c()`, `seq()`, `:`). Discuss the convenience of each method.

```r
# Generate a vector using c() method
vector_c <- c(100, 101, 102, 103, 104, 105)

# Generate a vector using seq() method
vector_seq <- seq(100, 105, by = 1)

# Generate a vector using : operator
vector_colon <- c(100:105)
```

Answer: *The first method, generating a vector using* `c()` *is convenient when you are only including a few elements in your sequence or there is no clear pattern. The second method, using* `seq`*, is convenient when the numbers follow a pattern but not necessarily just increment by. The third method is most convenient to generate numbers in a sequence increasing or decreasing by exactly 1.*

2.2

Generate a sequence of all even numbers between 0 and 100. Use the `seq()` function.

```r
# Generate a sequence of all even numbers between 0 and 100
even_seq <- seq(0, 100, by = 2)
even_seq
```

```
 [1]   0   2   4   6   8  10  12  14  16  18  20  22  24  26  28  30  32  34  36
[20]  38  40  42  44  46  48  50  52  54  56  58  60  62  64  66  68  70  72  74
[39]  76  78  80  82  84  86  88  90  92  94  96  98 100
```

```r
# Create a descending sequence of numbers from 100 to 1
desc_seq <- seq(100, 1, by = -1)
desc_seq
```

```
  [1] 100  99  98  97  96  95  94  93  92  91  90  89  88  87  86  85  84  83
 [19]  82  81  80  79  78  77  76  75  74  73  72  71  70  69  68  67  66  65
 [37]  64  63  62  61  60  59  58  57  56  55  54  53  52  51  50  49  48  47
 [55]  46  45  44  43  42  41  40  39  38  37  36  35  34  33  32  31  30  29
 [73]  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11
 [91]  10   9   8   7   6   5   4   3   2   1
```

2.3

Create a descending sequence from 100 to 1 and assign it to a variable. Use the `seq()` function.

```r
# Create a descending sequence of numbers from 100 to 1
desc_seq_decrease <- seq(100, 1, by = -1)
desc_seq_decrease
```

```
  [1] 100  99  98  97  96  95  94  93  92  91  90  89  88  87  86  85  84  83
 [19]  82  81  80  79  78  77  76  75  74  73  72  71  70  69  68  67  66  65
 [37]  64  63  62  61  60  59  58  57  56  55  54  53  52  51  50  49  48  47
 [55]  46  45  44  43  42  41  40  39  38  37  36  35  34  33  32  31  30  29
 [73]  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11
 [91]  10   9   8   7   6   5   4   3   2   1
```

Exercise 3: Data Generation and Basic Statistical Analysis

3.1

Generate a sample of 100 observations from a normal distribution with a mean of 10 and a standard deviation of 2. Use the `rnorm()` function.

```
sim_data <- rnorm(n = 100,
                  mean = 10,
                  sd = 2)
```

3.2

What are the 1st, 10th, and 100th elements of this `vector`?

```
sim_data[c(1, 10, 100)]
```

```
[1] 13.77940 12.12523 12.16928
```

3.3

Calculate the mean of this `vector`. How does this `sample` mean relate to the `population` mean (hint: population mean = 10) of the distribution?

```
mean(sim_data)
```

```
[1] 9.986522
```

Answer: *This is relatively close to, but not exactly, the population mean. This is because we are taking a random sample from the distribution.*

3.4

Calculate the difference between the `sample` mean and the `population` mean. Discuss the reason for the discrepancy.

```r
diff_100 <- mean(sim_data) - 10
```

3.5

Repeat steps 1 and 3 with a sample size of 10,000. Did the difference between the sample mean and the population mean decrease? Why?

```r
## simulate 10000 draws
sim_data_10000 <- rnorm(n = 10000,
                        mean = 10,
                        sd = 2)

mean_sim_data_10000 <- mean(sim_data_10000)

##
mean_sim_data_10000 - 10
```

```
[1] -0.03449191
```

Answer*: The difference between the sample mean and the population mean decreased as we increased our sample size. This is because as our sample size increases, the mean of the sample tends towards the population mean. This is a fundamental concept in statistics: the law of large numbers.*