MTH 3220 R Notes 10

11 Miscellaneous Functions for Searching and Comparing Vectors

11.1 Set Operations

- R has several functions that perform ***set operations***:

```
union()       # Union of two vectors x and y, consisting of all
              # the elements that are in at least one of x or y.
intersect()   # Intersection of two vectors x and y, consisting of all
              # the elements that are in both x and y.
setdiff()     # Set difference of two vectors x and y, consisting of
              # all the elements that are in x but not in y.
setequal()    # Returns TRUE or FALSE depending on whether the elements
              # of two vectors x and y are the same.
%in%          # Returns TRUE or FALSE depending on whether a value x
              # is an element of a vector y.  We could also use
              # is.element().
```

- `union()` takes two vectors `x` and `y`, and returns a vector consisting of all the elements that are in `x` *or* in `y`. For example:

```
union(x = 1:5, y = 4:8)

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

- `intersect()` takes two vectors `x` and `y`, and returns a vector consisting of all the elements that are in `x` *and* in `y`:

```
intersect(x = 1:5, y = 4:8)

## [1] 4 5
```

- `setdiff()` takes vectors `x` and `y`, and returns a vector consisting of all the elements that are in `x` *but not* in `y`:

```
setdiff(x = 1:5, y = 4:8)

## [1] 1 2 3
```

- `setequal()` takes two vectors `x` and `y`, and returns `TRUE` or `FALSE` depending on whether their elements are the same:

```
setequal(x = c(1, 2, 3), y = c(3, 1, 2))

## [1] TRUE
```

- `%in%` Returns `TRUE` or `FALSE` depending on whether a value `x` is an element of a vector `y`. For example:

```
3 %in% c(4, 6, 2, 3, 7, 9)

## [1] TRUE
```

The first argument can be a vector. For example:

```
c(3, 2) %in% c(4, 6, 2, 3, 7, 9)

## [1] TRUE TRUE
```

We could also do this using the `is.element()` function.

Section 11.1 Exercises

Exercise 1 Guess what the result of each of the following commands will be, then check your answers.

a)
```
union(x = 1:4, y = 3:6)
```

b)
```
intersect(x = 1:4, y = 3:6)
```

c)
```
setdiff(x = 1:4, y = 3:6)
```

Exercise 2 Guess what the result of each of the following commands will be, then check your answers.

a)
```
setequal(x = c(8, 4, 2), y = c(2, 8, 4))
```

b)
```
setequal(x = c(4, 7, 9), y = c(7, 9, 6))
```

c)
```
4 %in% c(1, 5, 2, 6, 8)
```

d)
```
4 %in% c(1, 4, 2, 6, 8)
```

e)
```
"b" %in%  c("b", "a", "b", "c", "a", "a", "c")
```

11.2 Using `unique()`, `duplicated()`, `identical()`, and `all.equal()`

- Here are two functions that are useful for finding the unique and duplicated values in a vector:

```
unique()      # Returns the unique values of a vector
duplicated()  # Determines which elements of a vector are duplicates
              # of preceding elements in the vector
```

- `unique()` returns the *unique* values in a vector. For example, here's a vector `x`:

```
x

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

The unique values in `x` are:

```
unique(x)

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

- `unique()` works with `"character"` vectors too. For example, here's a vector `char.vec`:

```
char.vec

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

```
unique(char.vec)

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

- `duplicated()` determines which values in a vector are duplicates of values that occurred earlier in the vector (i.e. at a lower index value). For example, using `x` from above:

```
duplicated(x)

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

A `TRUE` means the corresponding value in the vector has already appeared at least once earlier in the vector.

- When we check two vectors `x` and `y` for equality by typing `x == y`, the result is a `"logical"` *vector*.

If we need a *single* `"logical"` (`TRUE` or `FALSE`) value depending on whether or not the vectors are the same, we can use either of the functions:

```
identical()     # Are two vectors (exactly) identical?  Returns TRUE
                # or FALSE.
all.equal()     # Are two vectors identical (allowing for minor dif-
                # ferences due to roundoff error)?  Used with isTRUE().
```

- For example, consider the following vectors:

```
x <- c(1, 3, 4, 6, 13)
y <- c(1, 3, 4, 6, 13)
```

Because `x` and `y` are (exactly) the same, `identical()` returns `TRUE`:

```
identical(x, y)

## [1] TRUE
```

If `x` and `y` had the same values, but in a different order, `identical()` would return `FALSE`.

Likewise, because `x` and `z` below are different:

```
x <- c(1, 3, 4, 6, 13)
z <- c(1, 3, 4, 6, 11)
```

`identical()` returns `FALSE`:

```
identical(x, z)

## [1] FALSE
```

- Note that we could also test whether two vectors are (exactly) identical by typing:

```
all(x == z)

## [1] FALSE
```

- The only numbers that can be represented exactly in R are integers and fractions whose denominator is a power of 2. All other numbers are internally rounded. As a result, `==` and `identical()` sometimes return `FALSE` unexpectedly when applied to (`"double"`) numbers:

```
a <- sqrt(2)
a * a == 2

## [1] FALSE

identical(a * a, 2)

## [1] FALSE
```

For insight:

```
a * a - 2

## [1] 4.440892e-16

print(a * a, digits = 18)

## [1] 2.0000000000000004
```

To test for equality, allowing for slight differences due to round off error, use `all.equal()`.

```
all.equal(a * a, 2)

## [1] TRUE
```

The example above came from the **R FAQ page**, which has a section devoted to this issue:
`https://cran.r-project.org/doc/FAQ/R-FAQ.html#Why-doesn_0027t-R-think-these-numbers-are-equal_003f`

Section 11.2 Exercises

Exercise 3 Here's a vector `x`:

```
x <- c(4, 4, 5, 6, 6, 6)
```

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

a) `unique(x)`

b) `duplicated(x)`

Exercise 4 Here's are two vectors `x` and `y`:

```
x <- c(7, 8, 5, 10)
x <- c(7, 8, 5, 12)
```

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

```
identical(x, y)
```

Exercise 5 The only numbers that can be represented exactly in R are integers and fractions whose denominator is a power of 2. All other numbers are internally rounded.

a) Guess what the (surprising) result of the following command will be, then check your answer:

```
0.1 + 0.05 == 0.15
```

To see why, try typing:

```
0.15 - (0.1 + 0.05)
print(0.1 + 0.05, digits = 18)
```

b) To test for equality, allowing for slight differences due to round off error, we use `all.equal()`.

Guess what the result of the following command will be, then check your answer:

```
all.equal(0.1 + 0.05, 0.15)
```