Week 5 Module 4: The Special Value `NA`

CSCI E-5a: Introduction to R

Let's clear the global environment:

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

Module Overview and Learning Objectives

Hello! And welcome to Module 4: The Special Value `NA`.

In this module, we'll learn about another special value, denoted `NA`.

- In section 1, we'll meet the special value `NA`, and discuss some issues relating to missing values in our data.
- In section 2, we'll learn about the basics of working with `NA` and how to interpret expressions with this value.
- In section 3, we'll encounter some problems when we try to naively test an R object to see if it is a missing value, and we'll learn how to resolve this problem.
- In section 4, we'll see how to use the `sum()` and `mean()` functions with data containing missing values.

Once you've completed this module, you'll be able to:

- Explain the concept of the special value `NA`.
- Evaluate expressions that involve the special value `NA`.
- Test an R object to see if it is a missing value.
- Calculate the sum and sample mean of the non-missing values in a vector.

We'll meet one new built-in R function in this module:

- `is.na()`

All right! Let's get started by learning about the special value `NA`.

Section 1: NA and Missing Data

Main Idea: *Let's learn about the special value `NA`*

In this section, we'll meet the special value `NA`, and discuss some issues relating to missing values in our data.

In Lecture 2, we saw the special values `Inf`, `-Inf`, `NaN`, and `NULL`.

There is another special value that is somewhat different, and you'll encounter it frequently.

This is the value `NA`, which represents a missing value.

That means that you were supposed to observe a value for a particular variable, but for whatever reason you didn't.

Missing data values are philosophically problematic, because they aren't really a "value" at all – instead, they represent the *absence* of a value.

The theory of most statistical procedures is based on the assumption that you have complete data, but in practice we are often stuck with missing data, and we have to deal with it as best we can.

In this course we'll develop some simple ways to handle missing data, but they won't be perfect.

Really, there is no ideal solution to this difficulty, other than to not generate missing data in the first place.

There are many ways in which missing data can arise in practice, and in any given situation you have to think about the specifics of your problem to determine what's appropriate.

In some cases, we have little option but to simply ignore any observation that is missing whenever we perform a calculation with that variable.

For instance, suppose we are analyzing patient data, with one of the variables is the age of the patient.

Some people might not want to report that, and thus we have missing data.

In this case, we can't really use this particular observation in any analysis that involves age, because we have no idea what the patient's true age is.

In other cases, you really can do something with the missing values.

Consider how Gradescope records problem set submissions.

If a student submits a problem set and does not receive credit for any problem, Gradescope will record a score of 0.

However, if a student does not submit a problem set at all, Gradescope records this as a missing value.

That makes sense – the student didn't submit any work, so really the score should be considered as missing.

On the other hand, when it comes to calculate the final grade, of course I'm going to score any missing problem set as a 0.

In this case, it makes sense to replace the missing value with a 0, and then to use this value in all calculations.

In the previous example with patient age this would be a completely inappropriate thing to do, because you *know* that nobody on your study has an age of 0.

So this illustrates an important aspect of working with missing data: it all depends on the particulars of your project, and you'll have to take responsibility for deciding exactly what to do with missing values in your analysis.

All that we can do in our course is to learn the various techniques for handling missing values, and then you'll have to use your creativity and imagination to apply these methods to your own practice.

So those are some of the basic concepts of `NA` values and missing data in general.

Now let's look at how to evaluate expressions with `NA` values.

Section 2: Working with NA Values

> **Main Idea:** *We can evaluate expressions that contain NA values*

In this section, we'll learn the basics of working with NA and how to interpret expressions with this value.

R handles NA values in a very rational way.

For arithmetic operations, once an NA is introduced, then the entire expression evaluates to NA:

```
12 + NA
```

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

This makes a lot of sense, because we can't know the value of the sum unless we know the values of all the terms in the sum.

For logical values, it's a little more complicated.

For instance, this expression has to evaluate to NA:

```
TRUE & NA
```

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

Again, this makes a lot of sense, because in order to evaluate this logical conjunction we have to know the logical value of both terms.

But now consider this expression:

```
FALSE & NA
```

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

R was able to evaluate this expression, because the rule for logical conjunctions is that once one term evaluates to FALSE then the entire expression evaluates to FALSE.

So in this case it actually doesn't matter that one of the values is missing, because the logical value of the entire expression can still be determined.

Likewise, consider this expression:

```
FALSE | NA
```

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

This evaluates to NA because R cannot determine the value of the expression.

However, now consider the expression:

```
TRUE | NA
```

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

Now R *can* evaluate this expression, because the rule for logical disjunctions is that once one term evaluates to TRUE then the entire expression evaluates to TRUE.

So it doesn't matter that the second term is missing, as long as the first term is TRUE.

So that's how to work with NA values in expressions.

Now let's see how to find out if an R object has the value NA.

Exercise 4.1: Expressions with `NA`

What are the values of these expressions?

```
(7 <= NA) | (12 - 2 > 5)
```

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

```
(6 > NA) & (NA == 3) & (7 <= -5)
```

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

```
(TRUE & NA) | !(FALSE & TRUE)
```

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

Solution

Section 3: Testing for Missingness

> **Main Idea:** *We can test to see if an object has the value `NA`*

In this section, we'll learn how to test an R object to see if it is missing.

We run into problems if we want to test an R object to see if it is missing.

For instance, look at what happens with this code:

```
4 == NA
```

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

Wait a second!

The object on the left-hand side of this expression has the value 4, and that is certainly not the same as the special value `NA`.

So you might think that if we test the value 4 for equality with `NA`, it should return `FALSE`.

But instead R returned `NA`.

Why did this happen?

Think for a few moments and see if you can figure out what's going on here . . .

Did you get it?

When R evaluates the test for equality, it first evaluates the term on the left-hand side, and this of course evaluates to 4.

Then R attempts to evaluate the term on the right-hand side, but this is missing.

So R cannot perform the test for equality, because one of the terms doesn't have a value.

Essentially, we are asking R the question, "Is the value 4 equal to something for which I don't know the value?"

R very reasonably responds with "Well, if you don't know the value of the term on the right-hand side, then I can't determine the value of the test for equality, so I'm just going to return `NA`."

Earlier I said that missing data is "philosophically problematic", and this example illustrates the exact nature of the problem.

Even though I said that `NA` is a "special value", in fact it's not a value *at all* – instead, it represents the absence of a value.

Here's another example of this sort of thing, which is perhaps even weirder.

In this code, we are trying to test the special value 'NA' for equality with the special value `NA`.

```
NA == NA
```

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

How can something not be equal to itself?

You might think that this should return the value `TRUE`, because we are testing something for equality with itself, and surely the expression `x == x` should always be `TRUE`, regardless of what `x` represents.

But in fact R still returns the value `NA`, because now it can't determine *any* of the actual values.

In essence, we are asking R the question, "Is the value of something, which I don't know the value of, equal to something else that I also don't know the value of?"

R very understandably replies, "Well, if you don't know the values of either of the terms in the comparison test, then there is no way that I can determine whether it's `TRUE` or `FALSE`, so I'm just going to return `NA`".

Once again, the object `NA` isn't really a value – it represents the *absence* of a value.

Let's try this with a variable:

```
test <- NA

test == NA
```

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

The bottom line here is that if we want to test an R object to see if it is `NA` we can't use an equality comparison.

OK, so now we know that we can't treat `NA` as a regular value such as 9 or -2.562 or `TRUE`.

But we still would like to be able to test a value to determine whether or not its value is missing.

How do we do that?

The answer is: there's a built-in function for that.

We can use the `is.na()` function, which takes one input argument x and returns the logical value `TRUE` if the input argument `x` evaluates to `NA`, and `FALSE` otherwise.

For instance, let's try to determine if the value 4 is missing:

```
is.na( x = 4 )
```

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

That's much better!

Let's try it with the second example:

```
x <- NA

is.na( x )
```

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

Again, we are now getting the answer we want.

So the moral of this story is: when you want to test something to see if it's a missing value, don't try to use a comparison operator, but instead use the function `is.na()`.

I know what you're wondering.

"Sure, this function `is.na()` works with a single value, but can we *vectorize* it?"

Let's create a test vector with some `NA` values:

```
test.vector <-
    c(5, 7, 3, NA, 4, 1, NA, 9)
```

Now let's perform a vectorized test on this `test.vector`.

Can you predict what the result will be?

```
is.na( test.vector )
```

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

So the answer is yes, you can vectorize is.na()'.

Of course, once you've constructed this logical vector using `is.na()`, you can then apply all the logical functions that we've used before.

So that's how we can test for missingness.

Now let's consider how `NA` values operate when we calculate summary statistics.

Exercise 4.2: Working with `NA`

This is a very important technique.

Here's a vector that contains missing data:

```
exercise.4.2.data <-
    c(3, 5, 7, NA, 4, 9, NA, 3)
```

I want to answer three questions about this vector:

- Does the vector contain any missing data?
- How many missing elements does the vector contain?
- What are the locations of the missing data?

Hint: use logical functions and the function `is.na()`.

Answer

Section 4: Summary Statistics and `NA`

> **Main Idea:** *We need to be careful when calculating summary statistics with data that contains `NA` value*

In this section, we'll see how to use the `sum()` and `mean()` functions with data containing missing values.

Unfortunately, summary statistics functions such as `sum()` and `mean()` do not play nicely with `NA`.

Let's consider `test.vector` again:

```
test.vector
```

```
## [1]  5  7  3 NA  4  1 NA  9
```

What happens if we try to add up these values?

```
sum( test.vector )
```

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

Strictly speaking, this makes sense – some of the values in the vector are missing, so really we cannot determine what the sum of the values is, because we don't know them all.

This is also the case for the `mean()` function:

```
mean( test.vector )
```

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

Again, strictly speaking, this is probably the most "correct" response, because we can't calculate the mean of the values of `test.vector` if we don't know them all.

However, in practice we usually just want to ignore the missing values, and have R calculate the sum or the mean of the non-missing values.

We can do this by using the `na.rm = TRUE` option, which instructs R to ignore the missing values.

Let's calculate the sum of the non-missing values in `test.vector`:

```
sum( test.vector, na.rm = TRUE )
```

```
## [1] 29
```

When we use the `na.rm = TRUE` option with the `mean()` function, it excludes the missing values from the sum in the numerator and also the count in the denominator.

```
mean( test.vector, na.rm = TRUE )
```

```
## [1] 4.833333
```

Thus, when we use the `na.rm = TRUE` option with the `mean()` function, it effectively removes all the `NA` values first, and then calculates the sample mean in the usual way.

Personally, I think that this is an unfortunate choice for the convention, and R should ignore missing values by default, but it's too late now to change it.

So that's how to calculate summary statistics with data that contains `NA` values.

Now let's review what we've learned in this module.

Exercise 4.3: Summary statistics

Recall the vector from exercise 4.2:

```
exercise.4.2.data <-
    c(3, 5, 7, NA, 4, 9, NA, 3)
```

Calculate the sum of the non-missing values of this vector.

Then calculate the sample mean of the non-missing values of this vector.

Solution

Module Review

In this module, we learned about another special value, denoted `NA`.

- In section 1, we met the special value `NA`, and discussed some issues relating to missing values in our data.
- In section 2, we learned about the basics of working with `NA` and how to interpret expressions with this value.
- In section 3, we encountered some problems when we try to naively test an R object to see if it is a missing value, and we'll learned how to resolve this problem.
- In section 4, we saw how to use the `sum()` and `mean()` functions with data containing missing values.

Now that you've completed this module, you should be able to:

- Explain the concept of the special value `NA`.
- Evaluate expressions that involve the special value `NA`.
- Test an R object to see if it is a missing value.
- Calculate the sum and sample mean of the non-missing values in a vector.

We met one new built-in R function:

- `is.na()`

All right! That's it for Module 4: The Special Value `NA`.

Now let's move on to Module 5: Conditional Branching.

Solutions to the Exercises

Exercise 4.1: Expressions with `NA`

What are the values of these expressions?

```
(7 <= NA) | (12 - 2 > 5)
```

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

```
(6 > NA) & (NA == 3) & (7 <= -5)
```

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

```
(TRUE & NA) | !(FALSE & TRUE)
```

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

Solution

Exercise 4.2: Working with `NA`

This is a very important technique.

Here's a vector that contains missing data:

```
exercise.4.2.data <-
    c(3, 5, 7, NA, 4, 9, NA, 3)
```

I want to answer three questions about this vector:

- Does the vector contain any missing data?
- How many missing elements does the vector contain?
- What are the locations of the missing data?

Hint: use logical functions and the function `is.na()`.

Answer

To determine if the vector contains missing data, we have:

```
any( is.na( exercise.4.2.data ) )
```

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

To count the number of missing elements, we have:

```
sum( is.na( exercise.4.2.data ) )
```

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

The locations of the missing elements are:

```
which( is.na( exercise.4.2.data ) )
```

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

Exercise 4.3: Summary statistics

Recall the vector from problem 4.2:

```
exercise.4.2.data <-
    c(3, 5, 7, NA, 4, 9, NA, 3)
```

Calculate the sum of the non-missing values of this vector.

Then calculate the sample mean of the non-missing values of this vector.

Solution

The sum of the non-missing values is:

```
sum( exercise.4.2.data, na.rm = TRUE )
```

```
## [1] 31
```

The sample mean of the non-missing values is:

```
mean( exercise.4.2.data, na.rm = TRUE )
```

```
## [1] 5.166667
```