Week 3 Module 1: Vector Basics

CSCI E-5a: Programming in R

Let's clear the global environment:

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

Module Overview and Learning Outcomes

Hello! And welcome to Module 1: Vector Basics.

In this module, we will begin our in-depth study of vectors.

- In section 1, we'll see how to load R objects in from a file.
- In section 2, we'll define the concept of a vector.
- In section 3, we'll learn some useful functions when working with vectors.
- In section 4, we'll learn two useful summary statistics for numeric vectors.
- In section 5, we'll learn how to report vectors.

When you've finished this module, you'll be able to:

- Load in R objects from a file.
- Define vectors in terms of their two fundamental properties.
- Use the `head()`, `length()`, `sum()`, and `prod()` functions.
- Represent sets using vectors.
- Summarize numeric vectors using the `mean()`, `var()`, and `sd()` functions.
- Report a vector using the `cat()` and `formatC()` functions, along with the `head()` function and the `fill = TRUE` option.
- `load()`
- `head()`
- `length()`
- `sum()`
- `prod()`
- `mean()`
- `var()`
- `sd()`

Section 1: Loading R Objects

Main Idea: *We can load objects in from a file*

In this section, we'll see how to load R objects in from a file.

We can save R objects to a file, and then load them back in during a later session.

To load R objects into an R session, use the `load()` function.

The official documentation describes this function as "Reload saved datasets", but in fact it's much more general, and can load *any* type of R object, not just datasets.

The `load()` function takes one input argument, which is the name of the file containing the R objects.

For right now, the R object file should be in the same folder as the current R notebook.

Later on, we'll see how we can relax this requirement, using a technique known as a "path".

I recommend that you make your life simple and just be sure that the R object file is in the same folder as the R notebook.

If you look in the folder for this notebook, you should see an icon for a file named "Module 1 R Objects.Rdata".

We can check the contents of the folder containing the current R notebook by using the `dir()` function, which takes no input arguments:

```
dir()
```

```
## [1] "Module 1 R Objects.Rdata"
## [2] "Week_3_Module_1_Vector_Basics.nb.html"
## [3] "Week_3_Module_1_Vector_Basics.Rmd"
```

R object files have the file extension ".Rdata".

Let's load in the R objects contained in the file "Module 1 R Objects.Rdata".

Notice that right now the environment is empty.

Now we'll load in the R objects:

```
load( "Module 1 R Objects.Rdata" )
```

Notice that the environment is now populated with R objects.

The objects that have been loaded are the same as if we had created them ourselves during this session, and we can use them like any other R object.

```
test.vector
```

```
##  [1]  93.73546 101.83643  91.64371 115.95281 103.29508  91.79532 104.87429
##  [8] 107.38325 105.75781  96.94612 115.11781 103.89843  93.78759  77.85300
## [15] 111.24931  99.55066  99.83810 109.43836 108.21221 105.93901 109.18977
## [22] 107.82136 100.74565  80.10648 106.19826
```

When I send out a ZIP bundle, the R object files should all be in the correct location.

We'll see later how to create our own R object files.

Being able to save and load R objects is a powerful technique, and we'll use it throughout the course.

So that's how to load R objects in from a file.

Now let's learn how to define the concept of a vector in R.

Section 2: Defining Vectors

Main Idea: *We can define the concept of a vector in R*

In this section, we'll define the concept of a vector in R.

The most fundamental compound data structure in R is the *vector*, which is just an ordered sequence of values.

Vectors have two fundamental properties:

- First, all the values in the vector must be of the same atomic data type.
- Second, since a vector is an ordered sequence, it is a one-dimensional object.

The first condition is easy to understand – we can't mix and match different atomic data types within a single vector, but instead all the values must be of the same atomic type.

This code will store the vector into a variable called `all.numeric.values.vector`.

```
all.numeric.values.vector <-
    c( 5, -3, 7, 4, 2, 6, -2.5, 4 )
```

Now when we type in the variable name, R returns the vector:

```
all.numeric.values.vector
```

```
## [1]  5.0 -3.0  7.0  4.0  2.0  6.0 -2.5  4.0
```

We can also have a vector with only logical values:

```
all.logical.values.vector <-
    c( TRUE, TRUE, FALSE, TRUE, FALSE )

all.logical.values.vector
```

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

And we can even have a vector with all character values:

```
all.character.values.vector <-
    c( "To", "be", "or", "not", "to", "be" )

all.character.values.vector
```

```
## [1] "To"  "be"  "or"  "not" "to"  "be"
```

However, we can't have a vector with both numeric and logical values, and if we try to fool R into accepting such a structure it will automatically modify the values to all be consistent.

An important general rule to remember is that if a vector contains at least one object that cannot be interpreted as a number or a logical value, then it will convert everything in the vector into a character string.

Let's see an example of that:

```
c( 1, 4, 8, -6, "Bob", 0, 11 )
```

```
## [1] "1"   "4"   "8"   "-6"  "Bob" "0"   "11"
```

This is important when working with typos in numeric data, as we'll see later.

The second condition, that a vector is a "one-dimensional" object, is a little trickier, and we need to be careful here.

A vector is "one-dimensional" in the sense that when we print out the vector all the values are arranged along a line, and thus we can specify the position of a particular value using a single number.

For instance, in this vector, the value 7 occurs as the third item in the sequence, so we could specify this value by using the single number 3, indicating its position:

```
c(5, 2, 7)
```

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

This is potentially confusing, because we could also think of this vector as giving the coordinates of a point in three-dimensional space, in which case the vector could be considered as a three-dimensional object.

In fact, if you've had a course in linear algebra, this is exactly what you would think.

So, just to be clear: we *don't* think of a vector in this linear algebra way, and instead we always think of a vector as a one-dimensional object, no matter how many items are in the sequence.

The values in a vector are called the "elements" or the "components" or the "items" of the vector, and we say that these elements are "contained" in the vector.

Since a vector is an ordered sequence, we can also refer to the "position" or "location" of a particular element, which is just the number of the element in the sequence.

Thus, in this vector, the value of the element in the third position is 7:

```
all.numeric.values.vector
```

```
## [1]  5.0 -3.0  7.0  4.0  2.0  6.0 -2.5  4.0
```

You might have noticed that the value 4 occurs twice in this vector, in both the fourth and the last positions.

That's fine, and repeated values within a vector are possible.

In fact it's perfectly acceptable to have a vector that consists of just one value repeated over and over:

```
c(8, 8, 8, 8, 8, 8, 8, 8)
```

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

Let's summarize the ideas of this section:

- A vector is an ordered sequence of values.
- All the values must be of the same atomic data type.
- The values in the vector are called the "elements" or "components" of the vector.
- A vector is a one-dimensional structure, in the sense that we need only one number to specify the position of a particular element in the vector.

So that's we can define the concept of a vector in R.

Now let's learn about some useful vector functions.

Section 3: Vector Functions

Main Idea: *Let's learn some useful vector functions*

In this section, we'll learn some useful functions when working with vectors.

R has some special built-in functions that are useful when working with vectors.

The `head()` function

The `head()` function takes a vector of any class and an integer value n as input arguments, and returns the first n elements of the vector.

For instance, to directly display the first 5 values of the `all.numeric.values.vector`, we have:

```
head(
  x = all.numeric.values.vector,
  n = 5
)
```

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

The `length()` function

The `length()` function takes a vector of any class as its only input argument and returns the number of elements in the vector.

For instance, `all.numeric.values.vector` has 8 elements:

Thus, the `length()` function will return the value 8 when we use `all.numeric.values.vector` as the input argument:

```
length(
  x = all.numeric.values.vector
)
```

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

It's possible to manually change the length of a vector, but I think this is generally a bad idea, and we won't do it in this course.

The `sum()` function

The `sum()` function takes a numeric vector as its input argument, and returns the sum of all the elements of the vector.

```
sum(
  x = all.numeric.values.vector
)
```

```
## [1] 22.5
```

The `prod()` function

The `prod()` function takes a numeric vector as its input argument, and returns the product of all the elements of the vector.

```
prod(
  x = all.numeric.values.vector
)
```

```
## [1] 50400
```

So those are some special built-in functions that are useful when working with vectors.

Now let's learn how to summarize the values in a numeric vector.

Exercise 1.1: Exploring a vector

Display the first 8 elements of the `test.vector` that we loaded in from the R object file.

Then determine the length of the vector, as well as the sum and product of the elements of the vector.

Solution

```
head( test.vector, n = 6 )
```

```
## [1]  93.73546 101.83643  91.64371 115.95281 103.29508  91.79532
```

Section 4: Summary Statistics

Main Idea: *We can summarize the values in numeric vectors*

In this section, we'll learn three useful summary statistics for numeric vectors.

A *summary statistic* is a number that is calculated from observed data, and that can be used to summarize some general property of that data.

Three of the most important summary statistics are:

- The sample mean, which summarizes the overall average of the values in the vector.
- The sample variance, which summarizes how spread out are the values in a vector.
- The sample standard deviation, which also summarizes how spread out are the values in a vector.

R has built-in functions to perform these calculations.

The sample mean

The *sample mean* is undoubtedly the most important summary statistic.

The sample mean is calculated by adding all the elements together and then dividing by the number of elements:

$$\text{Sample mean} = \frac{\sum_{i=1}^{n} x_i}{n}$$

We can calculate the sample mean by hand by using the built-in R functions `sum()` and `length()`:

```
test.data <- c(4, 3, 8, 7, 2, 11, 6)

sample.mean <-
    sum( test.data ) / length( test.data )

cat(
    "The sample mean is:",
    formatC(
        sample.mean,
        format = "f",
        digits = 2
    )
)
```

```
## The sample mean is: 5.86
```

In practice, it's actually better not to calculate the sample mean by hand, but instead to use the built-in R function `mean()`:

```
sample.mean <-
    mean( test.data )

cat(
    "The sample mean is:",
    formatC(
        sample.mean,
        format = "f",
        digits = 2
    )
)
```

```
## The sample mean is: 5.86
```

We can think of the sample mean as providing a measure of the central location of the data.

The sample variance

The sample variance is calculated by summing the squared differences from the sample mean and then dividing by the number of elements minus 1:

$$\text{Sample variance} = \frac{\sum_{i=1}^{n} (x_i - \overline{x})^2}{n-1}$$

In general it's better not to calculate the sample variance by hand, but instead to use the built-in R function `var()`:

```
sample.variance <-
    var( test.data )

cat(
    "The sample variance is:",
    formatC(
        sample.variance,
        format = "f",
        digits = 2
    )
)
```

```
## The sample variance is: 9.81
```

The sample standard deviation

Because the variance is computed as a squared quantity, it can be difficult to understand.

In practice, it's often easier to think about the sample standard deviation, which is just the square root of the sample variance.

We can use the built-in R function `sd()` to calculate the standard deviation of the values in a numeric vector:

```
sample.standard.deviation <-
    sd( test.data )

cat(
    "The sample standard deviation is:",
    formatC(
        sample.standard.deviation,
        format = "f",
        digits = 2
    )
)
```

```
## The sample standard deviation is: 3.13
```

So that's how to summarize the values in a numeric vector.

Now let's see how to report vectors.

Exercise 1.2: Mean, variance, and standard deviation

Calculate the sample mean of the values in `test.vector`. Report your result using a `cat()` statement, displaying this value with 2 decimal places.

Then calculate the sample variance of the values in `test.vector`. Report your result using a `cat()` statement, displaying this value with 2 decimal places.

Finally, calculate the standard deviation of the values in `test.vector`. Report your result using a `cat()` statement, displaying this value with 2 decimal places.

Solution

Section 5: Reporting Vectors

Main Idea: *We can report vectors*

In this section, we'll learn how to report vectors.

In this course, we are careful about how we report our results.

In part that's because we want to be able to grade your work efficiently.

But more importantly it's because you ultimately want to be able to communicate your findings to an audience, and to do this you need to be able to report your results in a clear and understandable way.

Sometimes, your result must be reported in a very specific manner, and if you don't do it exactly right then you're just wrong.

We've already seen this with baseball statistics – we have to report Babe Ruth's batting average as 0.342, using exactly 3 decimal places.

It would be strange to report Babe Ruth's batting average as 0.3, using only one decimal place.

It would also be strange to report Babe Ruth's batting average as 0.3420645, using seven decimal places.

It all depends on the context, so we want you to know how to use the formatting tools.

The simplest way to report a vector is to just directly display it:

```
simple.vector <-
    c(5, 3, 6, 2)

simple.vector
```

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

It's OK to directly display a vector when we're in the middle of a computation and we just want to check the values very quickly.

But to more formally report your result, you should use a `cat()` statement:

```
cat(
    "simple.vector:",
    simple.vector
)
```

```
## simple.vector: 5 3 6 2
```

Usually I'll ask you to format these numbers to a certain number of decimal places, and for this you can use the `formatC()` function with the vector:

```
cat(
    "simple.vector:",
    formatC(
        simple.vector,
        format = "f",
        digits = 2
    )
)
```

```
## simple.vector: 5.00 3.00 6.00 2.00
```

Usually we don't need to see all the values of the vector, and we just want to see a few to check that it's correct.

In this case the `head()` function can be very useful.

I will often ask you to report a vector, and I will say something like, "Report the first 3 elements of `simple.vector` using a `cat()` statement, formatting the values with 2 decimal places."

Let's break this down:

- You will use the `cat()` function to print out an informative report.
- As always, you should indicate the name of the object that you are reporting.
- To obtain the first 3 items, use the `head()` function.
- To format the numbers with 2 decimal places, use the `formatC()` function.

This will be our standard method for reporting a vector using a `cat()` statement with specified formatting:

```
cat(
    "simple.vector:",
    formatC(
        head( x = simple.vector, n = 3 ),
        format = "f",
        digits = 2
    )
)
```

```
## simple.vector: 5.00 3.00 6.00
```

If you have a long vector, then the values will not all fit onto one line.

For instance, R has a built-in vector called `rivers`.

Let's directly display the `rivers` vector:

```
rivers
```

```
##   [1]  735  320  325  392  524  450 1459  135  465  600  330  336  280  315  870
##  [16]  906  202  329  290 1000  600  505 1450  840 1243  890  350  407  286  280
##  [31]  525  720  390  250  327  230  265  850  210  630  260  230  360  730  600
##  [46]  306  390  420  291  710  340  217  281  352  259  250  470  680  570  350
##  [61]  300  560  900  625  332 2348 1171 3710 2315 2533  780  280  410  460  260
##  [76]  255  431  350  760  618  338  981 1306  500  696  605  250  411 1054  735
##  [91]  233  435  490  310  460  383  375 1270  545  445 1885  380  300  380  377
## [106]  425  276  210  800  420  350  360  538 1100 1205  314  237  610  360  540
## [121] 1038  424  310  300  444  301  268  620  215  652  900  525  246  360  529
## [136]  500  720  270  430  671 1770
```

When we run a code chunk in RStudio, this vector is displayed nicely, with line breaks.

But when we knit to a PDF, it runs off the page.

The option `fill = TRUE` will insert line breaks when we knit to PDF:

```
cat( rivers, fill = TRUE )
```

```
## 735 320 325 392 524 450 1459 135 465 600 330 336 280 315 870 906 202 329 290
## 1000 600 505 1450 840 1243 890 350 407 286 280 525 720 390 250 327 230 265 850
## 210 630 260 230 360 730 600 306 390 420 291 710 340 217 281 352 259 250 470 680
## 570 350 300 560 900 625 332 2348 1171 3710 2315 2533 780 280 410 460 260 255
## 431 350 760 618 338 981 1306 500 696 605 250 411 1054 735 233 435 490 310 460
## 383 375 1270 545 445 1885 380 300 380 377 425 276 210 800 420 350 360 538 1100
## 1205 314 237 610 360 540 1038 424 310 300 444 301 268 620 215 652 900 525 246
## 360 529 500 720 270 430 671 1770
```

So that's how to report vectors.

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

Exercise 1.3: Report the `rivers` vector

Report the first 20 items of the built-in `rivers` vector, displaying these values using 2 decimal places.

Module Review

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

- Load in R objects from a file.
- Define vectors in terms of their two fundamental properties.
- Use the `head()`, `length()`, `sum()`, and `prod()` functions.
- Summarize numeric vectors using the `mean()`, `var()`, and `sd()` functions.
- Report a vector using the `cat()` and `formatC()` functions, along with the `head()` function and the `fill = TRUE` option.

Solutions to the Exercises

Exercise 1.1: Exploring a vector

Display the first 8 elements of the `test.vector` that we loaded in from the R object file.

Then determine the length of the vector, as well as the sum and product of the elements of the vector.

Solution

Let's display the first 8 elements of `test.vector`:

```
head( test.vector, 5 )
```

```
## [1]  93.73546 101.83643  91.64371 115.95281 103.29508
```

Next

```
prod( head( all.numeric.values.vector, 4 ) )
```

```
## [1] -420
```

Exercise 1.2: Mean, variance, and standard deviation

Calculate the sample mean of the values in `test.vector`. Report your result using a `cat()` statement, displaying this value with 2 decimal places.

Then calculate the sample variance of the values in `test.vector`. Report your result using a `cat()` statement, displaying this value with 2 decimal places.

Finally, calculate the standard deviation of the values in `test.vector`. Report your result using a `cat()` statement, displaying this value with 2 decimal places.

Solution

First, let's do the sample mean:

```
sample.mean <-
    mean( test.vector )

cat(
    "Sample mean:",
    formatC(
        sample.mean,
        format = "f",
        digits = 2
    )
)
```

```
## Sample mean: 101.69
```

Next, let's do the sample variance:

```
sample.variance <-
    var( test.vector )

cat(
    "Sample variance:",
    formatC(
        sample.variance,
        format = "f",
        digits = 2
    )
)
```

```
## Sample variance: 90.27
```

Finally, here's the sample standard deviation:

```
sample.standard.deviation <-
    sd( test.vector )

cat(
    "Sample standard deviation:",
    formatC(
        sample.standard.deviation,
        format = "f",
        digits = 2
    )
)
```

```
## Sample standard deviation: 9.50
```

Exercise 1.3: Report the `rivers` vector

Report the first 20 items of the built-in `rivers` vector, displaying these values using 2 decimal places.

Solution

```
cat(
    "rivers:",
    formatC(
        head( rivers, 20 ),
        format = "f",
        digits = 2
    ),
    fill = TRUE
)
```

```
## rivers: 735.00 320.00 325.00 392.00 524.00 450.00 1459.00 135.00 465.00 600.00
## 330.00 336.00 280.00 315.00 870.00 906.00 202.00 329.00 290.00 1000.00
```