Week 9 Module 3 – Working with Columns

CSCI E-5a: Programming in R

Let's clear the global computing environment:

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

Module Overview and Learning Objectives

Hello! And welcome to Module 3: Working with Columns.

In this module, we will investigate a variety of methods for selecting data from a data frame.

- In Section 1, we'll learn how to select columns from a data frame by using positive integer indexing.
- In Section 2, we'll see how to select columns from a data frame by using names.
- In Section 3, we'll see how to select individual elements from a data frame by using two identifiers.
- In Section 4, we'll learn how to sum the values in numeric columns or across rows.

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

- Select columns from a data frame by using positive integer indexing.
- Select columns from a data frame by using names.
- Select individual elements from a data frame by using two identifiers.

There are two new built-in R functions in this module:

- `colSums()`
- `rowSums()`

Allright, let's get rolling!

Section 1: Selecting Columns with Positive Integers

> **Main Idea:** *We can select columns from a data frame by using positive integer indexing*

We can select columns from a data frame by using positive integer indexing.

Positive integer indexing for data frames operates in a manner similar to positive integer indexing for vectors.

First, we can write the name of the data frame, followed by a single opening square bracket, then a positive integer, and finally a closing square bracket:

```
first.column.cars.data.frame <-
    cars[ 1 ]

head( first.column.cars.data.frame )
```

```
##   speed
## 1     4
## 2     4
## 3     7
## 4     7
## 5     8
## 6     9
```

This will return the first column of the `cars` data frame, but notice that the object that is being returned is a data frame:

```
class( first.column.cars.data.frame )
```

```
## [1] "data.frame"
```

If you want to extract the data in the first column of the data frame, but have it returned to you as a vector, you have to use double square brackets like this:

```
head( cars[[ 1 ]] )
```

```
## [1] 4 4 7 7 8 9
```

Let's check the clas of this object:

```
class( cars[[ 1 ]] )
```

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

You can also use a vector of positive integers to obtain multiple columns at once, although in this case the object that is returned must be a data frame, because we have multiple columns.

For instance, to obtain the first three columns of the `iris` data frame, we could use a colon operator to make a vector with the values 1, 2, and 3, and then use this to index the data frame:

```
first.three.iris.columns <- iris[ 1:3 ]

head( first.three.iris.columns )
```

```
##   Sepal.Length Sepal.Width Petal.Length
## 1          5.1         3.5          1.4
## 2          4.9         3.0          1.4
## 3          4.7         3.2          1.3
## 4          4.6         3.1          1.5
## 5          5.0         3.6          1.4
## 6          5.4         3.9          1.7
```

Likewise, if we only wanted the first and third columns, we could put the column indices 1 and 3 into a vector using the `c()` function:

```r
first.third.iris.columns <- iris[ c(1, 3) ]

head( first.third.iris.columns )
```

```
##   Sepal.Length Petal.Length
## 1          5.1          1.4
## 2          4.9          1.4
## 3          4.7          1.3
## 4          4.6          1.5
## 5          5.0          1.4
## 6          5.4          1.7
```

So that's how to select columns from a data frame by using positive integer indexing.

Now let's see how to select columns from a data frame by using names.

Exercise 2.1: Selecting columns with positive integer indexing

Write an expression that will extract all the columns to the left of `Sepal.Width`, including the `Sepal.Width` column as well.

Solution

```r
# Type your answer in here
```

Section 2: Selecting Columns with Names

> **Main Idea:** *We can select columns from a data frame by using their names*

In this section, we'll see how to select columns from a data frame by using names.

Since each column in a data frame has a name, we can also select columns by using their names.

Again, if we use a single square bracket, this will return the data in a data frame:

```r
head( cars[ "speed" ] )
```

```
##   speed
## 1     4
## 2     4
## 3     7
## 4     7
## 5     8
## 6     9
```

If we use double brackets, we will obtain the column in vector form:

```r
head( cars[[ "speed" ]] )
```

```
## [1] 4 4 7 7 8 9
```

We can also use a character vector to select multiple columns at once, bundled as a data frame:

```r
head( mtcars[ c("mpg", "cyl", "disp", "wt") ] )
```

```
##                    mpg cyl disp    wt
## Mazda RX4         21.0   6  160 2.620
## Mazda RX4 Wag     21.0   6  160 2.875
## Datsun 710        22.8   4  108 2.320
## Hornet 4 Drive    21.4   6  258 3.215
## Hornet Sportabout 18.7   8  360 3.440
## Valiant           18.1   6  225 3.460
```

There's another way to use names to obtain a column vector from a data frame: we can use the data frame name, followed by a dollar sign "$", and then the column name:

```r
head( iris$Sepal.Width )
```

```
## [1] 3.5 3.0 3.2 3.1 3.6 3.9
```

We could also construct the same R object by using the double bracket approach:

```r
head( iris[[ "Sepal.Width" ]] )
```

```
## [1] 3.5 3.0 3.2 3.1 3.6 3.9
```

So that's how to select columns from a data frame by using names.

Now let's see how to select individual elements from a data frame by using two identifiers.

Exercise 2.2: Selecting columns using names

Construct a data frame by selecting these variables from the `mtcars` data frame:

- `hp`
- `qsec`
- `gear`
- `carb`

Once you've selected these columns, save them in a variable. Then display the first five rows of the data frame.

Solution

Section 3: Selecting Elements Using Two Identifiers

> **Main Idea:** *We can select individual elements from a data frame*

In this section, we'll see how to select individual elements from a data frame by using two identifiers.

We can select a single element in a data frame by using two positive integers.

To do this, type the name of the data frame, followed by an open square bracket, followed by the index of the row, then a comma separator, then the index of the column, and finally a close square bracket.

Recall the `iris` data frame:

```
head( iris )
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Thus, to obtain the element in the sixth row and second column of the `iris` data frame, we have:

```
iris[ 6, 2 ]
```

```
## [1] 3.9
```

You can also use a vector for the indices:

```
iris[ c(3, 5, 6), 2 ]
```

```
## [1] 3.2 3.6 3.9
```

We can also use names in the double indexing technique. To select the fifth element in the `speed` column, we have:

```
iris[ 5, "Petal.Width" ]
```

```
## [1] 0.2
```

We can also do this using the dollar sign notation:

```
iris$Petal.Width[ 5 ]
```

```
## [1] 0.2
```

Because of this method of double-indexing, we can use our techniques for flow of control in a very interesting way.

For an example of this method, let's start by creating a data frame with some missing data in it:

```
missing.cars.data <- cars
missing.cars.data[ 4, 1 ] <- NA
missing.cars.data[ 17, 2 ] <- NA
missing.cars.data[ 38, 1 ] <- NA
```

Let's take a look at this data frame:

```
missing.cars.data
```

Let's try selecting one of the missing elements:

```
missing.cars.data[ 17, 2 ]
```

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

```
missing.cars.data[ 17, "dist" ]
```

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

Now we want to write a reporter to search for missing data over the entire data frame.

We can do this using a double `for` loop:

```
for( col.index in 1:ncol(missing.cars.data) ) {

    for( row.index in 1:nrow(missing.cars.data) ) {

        if ( is.na( missing.cars.data[ row.index, col.index ] ) ) {

            cat( "Found missing data!\n" )

        }
    }
}
```

```
## Found missing data!
## Found missing data!
## Found missing data!
```

Notice what this code is doing – it's scanning over the entire data frame by scanning across the columns first and for each column scanning down the rows.

Thus, this double `for` loop enables us to search through the entire data frame for missing data.

Of course, the message that it's printing out isn't very useful, but this is an opportunity to use your own imagination and creativity to think of how to improve this.

We could perform this scan another way.

First let's write a function that takes a vector and prints out a message if the vector contains any missing data:

```
vector.missing.data <- function( input.vector ) {
    if( any( is.na( input.vector ) ) ) {
        cat( "Found missing data!!\n" )
    }
}
```

Let's see how this function works on the first column of the `missing.cars` data frame:

```
vector.missing.data( missing.cars.data[[ 1 ]] )
```

```
## Found missing data!!
```

Notice that we had to use the double brackets notation, because we wanted the first column of the data frame as a vector.

Now we can apply this function to each column using a `for` loop:

```
for( col.index in 1:ncol( missing.cars.data ) ) {
    vector.missing.data( missing.cars.data[[ col.index ]] )
}
```

```
## Found missing data!!
## Found missing data!!
```

Again, this function is not as useful as it could be, and this is another opportunity for you to exercise your creativity and improve this code.

So that's how to select individual elements from a data frame.

Now let's learn how to sum the values in numeric columns or across rows.

Section 4: Summing Rows and Columns

Main Idea: *We can sum the values in numeric columns or rows*

In this section, we'll learn how to sum the values in numeric columns or across rows.

If we have a data frame consisting of all numeric values, then we can sum the values in each column or across the rows.

For instance, let's start by making a small test data frame to study:

```
test.data.frame <-
    data.frame(
        first = 1:3,
        second = 4:6,
        third = 7:9
    )

test.data.frame
```

```
##   first second third
## 1     1      4     7
## 2     2      5     8
## 3     3      6     9
```

We can construct a vector consisting of the sums of all the values in each column by using the `colSums()` function:

```
colSums( test.data.frame )
```

```
##  first second  third
##      6     15     24
```

Notice that the `colSums()` function returns a named vector, where the names are just the names of the columns in the original data frame.

We can also construct a vector consisting of the sums across the rows of this data frame by using the `rowSums()` function:

```
rowSums( test.data.frame )
```

```
## [1] 12 15 18
```

The `colSums()` and `rowSums()` functions can be very useful, but they necessarily require that all the values in the data frame are numeric.

If even one column in the data frame is not numeric, then the `colSums()` and `rowSums()` functions will generate an error.

To see this, let's construct another test data frame:

```
second.test.data.frame <-
    data.frame(
        name = c( "Taylor", "Ted", "Ashley" ),
        first = 1:3,
        second = 4:6,
        third = 7:9
    )

second.test.data.frame
```

```
##     name first second third
## 1 Taylor     1      4     7
## 2    Ted     2      5     8
## 3 Ashley     3      6     9
```

Now look what happens when we try to run the `colSums()` function with this data frame:

```
colSums( second.test.data.frame )
```

```
## Error in colSums(second.test.data.frame): 'x' must be numeric
```

The problem here is that the first column is not numeric, so R generates an error when it attempts to add character string values together.

A similar error occurs with the `rowSums()` function:

```
rowSums( second.test.data.frame )
```

```
## Error in rowSums(second.test.data.frame): 'x' must be numeric
```

The solution to this problem is to first select just the numeric columns that we want:

```
numeric.test.data.frame <-
    second.test.data.frame[
        c( "first", "second", "third" )
    ]

numeric.test.data.frame
```

```
##   first second third
## 1     1      4     7
## 2     2      5     8
## 3     3      6     9
```

Now we can use the `colSums()` function without generating an error:

```
colSums( numeric.test.data.frame )
```

```
##  first second  third
##      6     15     24
```

Similarly for the row sums:

```
rowSums( numeric.test.data.frame )
```

```
## [1] 12 15 18
```

This method of first selecting just the numeric columns makes the `colSums()` and `rowSums()` functions much more useful.

So that's how to sum the values in numeric columns or across rows.

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

Module Review

In this module, we investigated a variety of methods for selecting data from a data frame.

- In Section 1, we learned how to select columns from a data frame by using positive integer indexing.
- In Section 2, we saw how to select columns from a data frame by using names.
- In Section 3, we saw how to select individual elements from a data frame by using two identifiers.
- In Section 4, we learned how to sum the values in numeric columns or across rows.

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

- Select columns from a data frame by using positive integer indexing.
- Select columns from a data frame by using names.
- Select individual elements from a data frame by using two identifiers.

There were two new built-in R functions in this module:

- `colSums()`
- `rowSums()`

Allright, that's it for Module 3: Working with Columns.

Now let's move on to Module 4: Scatterplots.

See you there!

Solutions to the Exercises

Exercise 2.1: Selecting columns with positive integer indexing

Write an expression that will extract all the columns to the left of `Sepal.Width`, including the `Sepal.Width` column as well.

Solution

Here's a simple solution:

```
head( iris[ c(1, 2) ] )
```

```
##   Sepal.Length Sepal.Width
## 1          5.1         3.5
## 2          4.9         3.0
## 3          4.7         3.2
## 4          4.6         3.1
## 5          5.0         3.6
## 6          5.4         3.9
```

Here's another simple solution:

```
head( iris[ 1:2 ] )
```

```
##   Sepal.Length Sepal.Width
## 1          5.1         3.5
## 2          4.9         3.0
## 3          4.7         3.2
## 4          4.6         3.1
## 5          5.0         3.6
## 6          5.4         3.9
```

Here's a more abstract approach:

```
head(
    iris[ 1: which( names(iris) == "Sepal.Width" ) ]
)
```

```
##   Sepal.Length Sepal.Width
## 1          5.1         3.5
## 2          4.9         3.0
## 3          4.7         3.2
## 4          4.6         3.1
## 5          5.0         3.6
## 6          5.4         3.9
```

We'll find this useful when we want to write function that can perform this operation more generally.

Exercise 2.2: Selecting columns using names

Construct a data frame by selecting these variables from the `mtcars` data frame:

- `hp`
- `qsec`
- `gear`
- `carb`

Once you've selected these columns, save them in a variable. Then display the first five rows of the data frame.

Solution

```
exercise.2.2.data.frame <-
    mtcars[ c( "hp", "qsec", "gear", "carb" ) ]
```

Now we can display the first five rows using the `head()` function:

```
head( exercise.2.2.data.frame, 5 )
```

```
##                    hp  qsec gear carb
## Mazda RX4         110 16.46    4    4
## Mazda RX4 Wag     110 17.02    4    4
## Datsun 710         93 18.61    4    1
## Hornet 4 Drive    110 19.44    3    1
## Hornet Sportabout 175 17.02    3    2
```