Week 10 Module 1: Selecting, Sorting, and Creating

CSCI E-5a: Programming in R

Let's clear the global computing environment:

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

Module Overview And Learning Outcomes

Hello! And welcome to Module 1: Sorting, Selecting, and Creating.

In this module we'll investigate some basic techniques for managing the rows of a data frame.

- In Section 1, we'll see how to select rows from a data frame.
- In Section 2, we'll learn how to sort the rows of a data frame.
- In Section 3, we'll see how to create new columns in a data frame.

At the end of this module, you'll be able to:

- Select rows from a data frame
- Sort the rows of a data frame
- Create new columns in a data frame

There is one new built-in R function in this module:

- `order()`

All right! Let's start out by seeing how to select rows from a data frame.

Section 1: Selecting Rows

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

In this section, we'll see how to select rows from a data frame.

So far, all of our indexing methods have operated over columns.

There is a special notation that allows us to select rows.

To see how this works, let's create a simple data frame:

```
simple.data.frame <-
    data.frame(
        city =
            c("Boston", "Boston", "Boston",
                 "New York", "New York", "New York"),

        name =
            c("Anita", "Steve", "Lucy",
                 "Bob", "Taylor", "Steve")
    )

head( simple.data.frame )
```

```
##       city   name
## 1   Boston  Anita
## 2   Boston  Steve
## 3   Boston   Lucy
## 4 New York    Bob
## 5 New York Taylor
## 6 New York  Steve
```

Now we can select the third row by using positive integer indexing with a single set of square brackets.

However, there is a small wrinkle, in that to obtain all the columns for a row we have to use a comma after the row index, like this:

```
simple.data.frame[ 3, ]
```

```
##     city name
## 3 Boston Lucy
```

Of course, we can select multiple rows by using a numeric vector:

```
simple.data.frame[ c(1, 3), ]
```

```
##     city  name
## 1 Boston Anita
## 3 Boston  Lucy
```

Don't forget that weird little comma.

You can also use negative integer indexing:

```
simple.data.frame[ -2, ]
```

```
##       city   name
## 1   Boston  Anita
## 3   Boston   Lucy
## 4 New York    Bob
## 5 New York Taylor
## 6 New York  Steve
```

Again, don't forget that weird comma notation.

So far, we've seen how to select rows based on the row index.

But more often, you'll want to select rows based on some condition, and for this we can use logical indexing.

For instance, to select all the rows where the value of `city` is "New York", we have:

```
simple.data.frame[
    simple.data.frame$city == "New York", ]
```

```
##       city   name
## 4 New York    Bob
## 5 New York Taylor
## 6 New York  Steve
```

Do you see how this works?

First, we construct a logical vector which has the value `TRUE` in exactly those locations where `city` has the value "New York".

Nest, we use this logical vector to filter the rows of `simple.data.frame`.

As always, we have to use the comma after the row condition in order to obtain all the columns for these rows.

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

Now let's see how to sort the rows of a data frame.

Exercise 2.1

Create a data frame consisting of the rows of `simple.data.frame` that have "Steve" for a name.

Solution

```
# Type your answer in here.
```

Section 2: Sorting the Rows

> **Main Idea:** *We can sort the rows of a data frame*

In this section, we'll learn how to sort the rows of a data frame.

Often, we want to sort the rows of a data frame by the values in a particular column.

The `order()` function takes a vector and returns the sequence of indices of the ordered vector:

```
unsorted.vector <- c(4, 7, 1, 3, 6, 9, 2)

order( unsorted.vector )
```

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

Then we can sort the vector by using this `order()` along with positive integer indexing:

```
unsorted.vector[ order( unsorted.vector ) ]
```

```
## [1] 1 2 3 4 6 7 9
```

We can also sort in descending order by using the `decreasing = TRUE` option:

```
order( unsorted.vector, decreasing = TRUE )
```

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

```
unsorted.vector[
    order( unsorted.vector, decreasing = TRUE )
    ]
```

```
## [1] 9 7 6 4 3 2 1
```

We don't actually have to do this when we work with a vector, because R has a built-in function for sorting vectors, named `sort()`:

```
sort( unsorted.vector )
```

```
## [1] 1 2 3 4 6 7 9
```

However, it's a different matter when we have a data frame and we want to sort the rows by the values in some column.

For instance, let's say that we want to want to sort the rows of the `iris` data frame by the values in the `Sepal.Length` column.

First, let's quickly refresh our memory of the `iris` dataframe:

```
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
```

We can sort the `Sepal.Length` vector using our curious `order()` function:

```
iris$Sepal.Length[ order( iris$Sepal.Length ) ]
```

```
##   [1] 4.3 4.4 4.4 4.4 4.5 4.6 4.6 4.6 4.6 4.7 4.7 4.8 4.8 4.8 4.8 4.8 4.9 4.9
##  [19] 4.9 4.9 4.9 4.9 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.1 5.1 5.1 5.1
##  [37] 5.1 5.1 5.1 5.1 5.1 5.2 5.2 5.2 5.2 5.3 5.4 5.4 5.4 5.4 5.4 5.4 5.5 5.5
##  [55] 5.5 5.5 5.5 5.5 5.5 5.6 5.6 5.6 5.6 5.6 5.6 5.7 5.7 5.7 5.7 5.7 5.7 5.7
##  [73] 5.7 5.8 5.8 5.8 5.8 5.8 5.8 5.8 5.9 5.9 5.9 6.0 6.0 6.0 6.0 6.0 6.0 6.1
##  [91] 6.1 6.1 6.1 6.1 6.1 6.2 6.2 6.2 6.2 6.3 6.3 6.3 6.3 6.3 6.3 6.3 6.3 6.3
## [109] 6.4 6.4 6.4 6.4 6.4 6.4 6.4 6.5 6.5 6.5 6.5 6.5 6.6 6.6 6.7 6.7 6.7 6.7
## [127] 6.7 6.7 6.7 6.7 6.8 6.8 6.8 6.9 6.9 6.9 6.9 7.0 7.1 7.2 7.2 7.2 7.3 7.4
## [145] 7.6 7.7 7.7 7.7 7.7 7.9
```

Now we can use this to sort all the rows of the dataframe (don't forget the comma!):

```
head( iris[ order( iris$Sepal.Length ), ], n = 8 )
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 14          4.3         3.0          1.1         0.1  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 39          4.4         3.0          1.3         0.2  setosa
## 43          4.4         3.2          1.3         0.2  setosa
## 42          4.5         2.3          1.3         0.3  setosa
## 4           4.6         3.1          1.5         0.2  setosa
## 7           4.6         3.4          1.4         0.3  setosa
## 23          4.6         3.6          1.0         0.2  setosa
```

We can also sort the rows of the `iris` data frame by decreasing order of `Sepal.Length`:

```
head(
    iris[
        order( iris$Sepal.Length, decreasing = TRUE ), ] )
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 132          7.9         3.8          6.4         2.0 virginica
## 118          7.7         3.8          6.7         2.2 virginica
## 119          7.7         2.6          6.9         2.3 virginica
## 123          7.7         2.8          6.7         2.0 virginica
## 136          7.7         3.0          6.1         2.3 virginica
## 106          7.6         3.0          6.6         2.1 virginica
```

We can also perform a secondary sort on another variable, for instance `Petal.Length`:

```
head( iris[ order( iris$Sepal.Length, iris$Petal.Length ), ], 10 )
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 14          4.3         3.0          1.1         0.1  setosa
## 39          4.4         3.0          1.3         0.2  setosa
## 43          4.4         3.2          1.3         0.2  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 42          4.5         2.3          1.3         0.3  setosa
## 23          4.6         3.6          1.0         0.2  setosa
## 7           4.6         3.4          1.4         0.3  setosa
## 48          4.6         3.2          1.4         0.2  setosa
## 4           4.6         3.1          1.5         0.2  setosa
## 3           4.7         3.2          1.3         0.2  setosa
```

So that's how to sort the rows of a data frame.

Now let's see how to create new columns in a data frame.

Exercise 2.2

Sort the `cars` data frame in ascending order on the `dist` variable.

Solution

```
# Write your solution here
```

Section 3: Creating New Columns in a Data Frame

Main Idea: *We can create new columns in a data frame*

In this section, we'll see how to create new columns in a data frame.

Often, we'll want to create a new variable in a data frame based on other variables, often column vectors in the same data frame.

Recall that when we want to create a new variable for a numeric value, we simply use the variable name on the left-hand side of an assignment statement:

```
xyzyx <- 7.2
```

There was no variable `xyzyx` before we ran the assignment statement, but since it's on the left-hand side of the assignment operator R will create the variable for us.

The same thing is true for columns in a data frame: we can create new columns simply by using them on the left-hand side of an assigment statement.

For instance, suppose we have a data frame with information about the revenues and costs of four projects:

```
revenues.costs.data.frame <-
    data.frame(
        revenues = c(1000, 5000, 1200, 4000),
        costs = c( 600, 2200, 1400, 3100 )
    )

head( revenues.costs.data.frame )
```

```
##   revenues costs
## 1     1000   600
## 2     5000  2200
## 3     1200  1400
## 4     4000  3100
```

Let's create a new variable named `profit` that contains the profit for each project.

Remember that the profit is the revenue minus the cost.

Then we can create a new column consisting of the profits by simply using it on the left-hand side of an assignment operation:

```
revenues.costs.data.frame$profits <-
    revenues.costs.data.frame$revenues -
    revenues.costs.data.frame$costs

revenues.costs.data.frame
```

```
##   revenues costs profits
## 1     1000   600     400
## 2     5000  2200    2800
## 3     1200  1400    -200
## 4     4000  3100     900
```

Notice here that the new variable is on the left-hand side of an assignment statement.

When this occurs, R will automatically create a new column in the data frame to store this calculated value.

Of course, the new column still has to adhere to all the restrictions of a data frame:

- All the values must be of the same atomic data type.
- The length of the new column vector must be the same as the length of the other column vectors in the data frame.

So that's how to create new columns in a data frame.

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

Exercise 2.3

Suppose you have a data frame consisting of cost and revenues for a series of business transactions:

Costs	Revenues
1000	700
2000	2400
5000	6500
4000	4000

Create a data frame that represents this data. Then create two new rows:

- First, the profit for each project, defined as the revenues minus the costs.
- Second, the ratio of the revenues to the costs.

Solution

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

Module Review

In this module we investigated some basic techniques for managing data.

- In Section 1, we saw how to select rows from a data frame.
- In Section 2, we learned how to sort the rows of a data frame.
- In Section 3, we saw how to create new columns in a data frame.

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

- Select rows from a data frame.
- Sort the rows of a data frame.
- Create new columns in a data frame.

There was one new built-in R function in this module:

- `order()`

Solutions to the Exercises

Exercise 2.1

Create a data frame consisting of the rows of `simple.data.frame` that have “Steve” for a name.

Solution

```
subset.data.frame <-
    simple.data.frame[ simple.data.frame$name == "Steve", ]

head( subset.data.frame )
```

```
##       city  name
## 2   Boston Steve
## 6 New York Steve
```

Exercise 2.2

Sort the `cars` data frame in ascending order on the `dist` variable.

Solution

```
head( cars[ order( cars$dist ), ], n = 8 )
```

```
##    speed dist
## 1      4    2
## 3      7    4
## 2      4   10
## 6      9   10
## 12    12   14
## 5      8   16
## 10    11   17
## 7     10   18
```

Exercise 2.3

Suppose you have a data frame consisting of cost and revenues for a series of business transactions:

Costs	Revenues
1000	700
2000	2400
5000	6500
4000	4000

Create a data frame that represents this data. Then create two new rows:

- First, the profit for each project, defined as the revenues minus the costs.
- Second, the ratio of the revenues to the costs.

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

```
example.data.frame <-
    data.frame(
        costs = c(1000, 2000, 5000, 4000),
        revenues = c(700, 2400, 6500, 4000)
    )

example.data.frame$profits <-
    example.data.frame$revenues - example.data.frame$costs

example.data.frame
```

```
##   costs revenues profits
## 1  1000      700    -300
## 2  2000     2400     400
## 3  5000     6500    1500
## 4  4000     4000       0
```

```
example.data.frame$ratio.revenues.to.costs <-
    example.data.frame$revenues / example.data.frame$costs

example.data.frame
```

```
##   costs revenues profits ratio.revenues.to.costs
## 1  1000      700    -300                     0.7
## 2  2000     2400     400                     1.2
## 3  5000     6500    1500                     1.3
## 4  4000     4000       0                     1.0
```

Here's my solution:

```
example.data.frame <-
    data.frame(
        costs = c(1000, 2000, 5000, 4000),
        revenues = c(700, 2400, 6500, 4000)
    )

head( example.data.frame )
```

```
##   costs revenues
## 1  1000      700
## 2  2000     2400
## 3  5000     6500
## 4  4000     4000
```

Now we can create the profit variable:

```
example.data.frame$profit <-
    example.data.frame$revenues - example.data.frame$costs

head( example.data.frame )
```

```
##   costs revenues profit
## 1  1000      700   -300
## 2  2000     2400    400
## 3  5000     6500   1500
## 4  4000     4000      0
```

And now we can calculate the ratio of revenues to costs:

```
example.data.frame$revenues.costs.ratio <-
    example.data.frame$revenues / example.data.frame$costs

head( example.data.frame )
```

```
##   costs revenues profit revenues.costs.ratio
## 1  1000      700   -300                  0.7
## 2  2000     2400    400                  1.2
## 3  5000     6500   1500                  1.3
## 4  4000     4000      0                  1.0
```