Week 9 Module 2: Binding Rows and Columns

CSCI E-5a: Programming in R

Let's clear the global computing environment:

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

Module Overview and Learning Outcomes

Hello! And welcome to Module 2: Binding Rows and Columns.

In this module, we'll learn how to combine the rows or columns of data frames.

- In Section 1, we'll see how to append rows to a data frame.
- In Section 2, we'll see how to bind columns to a data frame.

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

- Combine two data frames together by binding rows, and explain the conditions that must be satisfied to perform this operation.
- Combine two data frames together by binding columns, and explain the conditions that must be satisfied to perform this operation.

There are 2 new built-in R functions in this module:

- `rbind()`
- `cbind()`

All right! Let's get started by learning how to append rows to a data frame.

Section 1: Appending Rows to a Data Frame

> **Main Idea:** *We can append rows to a data frame*

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

We can append rows to a data frame by using the `rbind()` function.

This function takes two arguments, both of which can be data frames or vectors.

The function returns a new data frame, in which the rows of the second input argument have been bound to the end of the first input argument.

When rows are attached to the end of a data frame, this is called "appending" the rows to the data frame.

The rule for `rbind()` is that the column names have to be the same, although they don't have to be in the same order.

Let's try a few experiments with some small data frames.

Here's the first data frame:

```
rbind.test.data.frame.1 <-
    data.frame(
        var.1 = 1:5,
        var.2 = 11:15,
        var.3 = 21:25
    )

rbind.test.data.frame.1
```

```
##   var.1 var.2 var.3
## 1     1    11    21
## 2     2    12    22
## 3     3    13    23
## 4     4    14    24
## 5     5    15    25
```

Here's the second data frame:

```
rbind.test.data.frame.2 <-
    data.frame(
        var.1 = 6:10,
        var.2 = 16:20,
        var.3 = 26:30
    )

rbind.test.data.frame.2
```

```
##   var.1 var.2 var.3
## 1     6    16    26
## 2     7    17    27
## 3     8    18    28
## 4     9    19    29
## 5    10    20    30
```

Now we can bind the rows of these data frames together:

```
rbind(
    rbind.test.data.frame.1,
    rbind.test.data.frame.2
)
```

```
##    var.1 var.2 var.3
## 1      1    11    21
## 2      2    12    22
## 3      3    13    23
```

```
## 4      4    14    24
## 5      5    15    25
## 6      6    16    26
## 7      7    17    27
## 8      8    18    28
## 9      9    19    29
## 10    10    20    30
```

The `rbind()` function uses the names of the columns to perform the matching, not the order of the columns.

Thus the columns in the two data frames do **NOT** have to be in the same order, as long as they have the same names.

Here's another data frame that is the same as `rbind.test.data.frame.2`, except that the columns are ordered differently:

```
rbind.test.data.frame.2.reordered <-
    data.frame(
        var.3 = 26:30,
        var.1 = 6:10,
        var.2 = 16:20
    )

rbind.test.data.frame.2.reordered
```

```
##   var.3 var.1 var.2
## 1    26     6    16
## 2    27     7    17
## 3    28     8    18
## 4    29     9    19
## 5    30    10    20
```

Then we have:

```
rbind(
    rbind.test.data.frame.1,
    rbind.test.data.frame.2.reordered
)
```

```
##    var.1 var.2 var.3
## 1      1    11    21
## 2      2    12    22
## 3      3    13    23
## 4      4    14    24
## 5      5    15    25
## 6      6    16    26
## 7      7    17    27
## 8      8    18    28
## 9      9    19    29
## 10    10    20    30
```

Notice that the order of the columns in the final data frame is the same as the order of the columns in the first input argument.

If we use `rbind()` with the `rbind.test.data.frame.2.reordered` data frame as the first input we'll get a different ordering of the columns:

```
rbind(
    rbind.test.data.frame.2.reordered,
    rbind.test.data.frame.1
)
```

```
##    var.3 var.1 var.2
## 1     26     6    16
## 2     27     7    17
## 3     28     8    18
## 4     29     9    19
## 5     30    10    20
## 6     21     1    11
## 7     22     2    12
## 8     23     3    13
## 9     24     4    14
## 10    25     5    15
```

If the column names are not the same, then `rbind()` won't be able to match the columns.

Look at this third data frame:

```
rbind.test.data.frame.3 <-
    data.frame(
        var.1 = 6:10,
        var.2 = 16:20,
        xyz = 26:30
    )

rbind.test.data.frame.3
```

```
##   var.1 var.2 xyz
## 1     6    16  26
## 2     7    17  27
## 3     8    18  28
## 4     9    19  29
## 5    10    20  30
```

Then we have:

```
rbind(
    rbind.test.data.frame.1,
    rbind.test.data.frame.3
)
```

```
## Error in match.names(clabs, names(xi)): names do not match previous names
```

What can we do if we have two data frames with different column names, but we want to bind the rows together?

We can use the `names()` function to adjust the names of the data frame columns so that everything is consistent.

In the data frame `rbind.test.data.frame.3`, we can change the column name of the third column from `xyz` to `var.3`:

```
names( rbind.test.data.frame.3 )[ 3 ] <- "var.3"
```

Let's check this:

```
rbind.test.data.frame.3
```

```
##   var.1 var.2 var.3
## 1     6    16    26
## 2     7    17    27
## 3     8    18    28
## 4     9    19    29
## 5    10    20    30
```

Now we can bind the two data frames together:

```
rbind( rbind.test.data.frame.1, rbind.test.data.frame.3)
```

```
##    var.1 var.2 var.3
## 1      1    11    21
## 2      2    12    22
## 3      3    13    23
## 4      4    14    24
## 5      5    15    25
## 6      6    16    26
## 7      7    17    27
## 8      8    18    28
## 9      9    19    29
## 10    10    20    30
```

So that's how to bind the rows of two data frames together.

Now let's see how to bind the columns of two data frames together.

Section 2: Binding Columns

> **Main Idea:** *We can bind columns to a data frame*

In this section, we'll see how to bind columns to a data frame.

Just as we can use the `rbind()` function to bind new rows to a data frame, we can use the `cbind()` function to bind new columns to the data frame.

You should be clear about the differences between these two procedures:

- With `rbind()`, we are adding a set of new observations to a pre-existing data frame, so the variables of the new observations have to match the variables of the pre-existing data frame.
- With `cbind()`, we are adding a set of new variables to a pre-existing data frame, so these rows of the new variables have to be in the same order as the rows of the pre-existing data frame.

Let's take a look at a simple example of how this works.

We'll start by creating a small data frame, called `cbind.example.data.frame`:

```
cbind.example.data.frame <-
    data.frame(
        first = 11:15,
        second = 21:25
    )

cbind.example.data.frame
```

```
##   first second
## 1    11     21
## 2    12     22
## 3    13     23
## 4    14     24
## 5    15     25
```

Let's also make a simple vector:

```
cbind.example.vector <-
    c(17, 23, 96, -8, 35)

cbind.example.vector
```

```
## [1] 17 23 96 -8 35
```

Now we can bind the vector to the data frame to make a new data frame with additional column:

```
cbind.data.frame <-
    cbind(
        cbind.example.data.frame,
        cbind.example.vector
    )

cbind.data.frame
```

```
##   first second cbind.example.vector
## 1    11     21                   17
## 2    12     22                   23
## 3    13     23                   96
## 4    14     24                   -8
## 5    15     25                   35
```

We were successful in binding the vector to the data frame, but the name of the third column is awkward. Let's call the `cbind()` function again, this time binding the name "third" to the vector:

```
cbind.data.frame <-
    cbind(
        cbind.example.data.frame,
        third = cbind.example.vector
    )

cbind.data.frame
```

```
##   first second third
## 1    11     21    17
## 2    12     22    23
## 3    13     23    96
## 4    14     24    -8
## 5    15     25    35
```

Much better!

You have to be very careful when you use the `cbind()` function.

To see why, recall that one of the fundamental properties of a data frame is that all the data in a row is somehow associated with the same "experimental unit".

Thus, all the data in the first row of a data frame is somehow associated with the first experimental unit, all the data in the second row is somehow associated with the second experimental unit, and so on.

That means that when we bind a vector to a data frame, all the data in the *vector* has to be consistent with the data frame ordering of experimental units.

If the vector is somehow out of order with the data frame, then the data will corrupted.

So you really have to sure that everything is lined up properly!

There are some cases where we really can be sure that everything lines up properly, and in those situations it's appropriate to use the `cbind()` function.

One example of where `cbind()` is the very useful is if we want to add in a simple `row.index` value to each row, so that for instance the row with a `row.index` value of 5 represents the measurements on the fifth experimental unit.

Sometimes it's convenient to have such an `row.index` variable, so that we can easily refer to particular experimental units, but many datasets do not have such an index.

Let's look at the built-in R data frame named `cars`:

```
head( cars )
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

You can see that R includes the row index on the left-hand column of the display, but these values are not actually part of the data.

Let's construct a new column in the data frame consisting of the row indices.

First, we know that the `cars` data frame has 50 rows:

```
nrow( cars )
```

```
## [1] 50
```

So our `row.index` variable should go from 1 to 50.

How can we make a vector that goes from 1 to 50 in steps of 1?

We can use the colon operator:

```
row.index.vector <- 1:50
```

Now we can bind that to the `cars` data frame:

```
indexed.cars.data.frame <-
    cbind( row.index = row.index.vector, cars )

head( indexed.cars.data.frame )
```

```
##   row.index speed dist
## 1         1     4    2
## 2         2     4   10
## 3         3     7    4
## 4         4     7   22
## 5         5     8   16
## 6         6     9   10
```

We can even define a new function called `create.row.index()` to perform this operation for any data frame, but we have to make a small adjustment to our code.

Before, we knew that we could just use 50 as the number of rows, because we were working with the `cars` data frame, and we knew that that data frame has exactly 50 rows.

But if we want to write a function that works for *any* input data frame, then we can't assume that there are exactly 50 rows any more.

Instead, we have to determine exactly how many rows there are using the `nrow()` function:

```
create.row.index <- function( input.data.frame ) {

    # First, have to find out the total number of rows in the data frame

    number.rows <- nrow( input.data.frame )

    # Next, we'll create the row.index.vector, using the number of rows that
    # we just calculated:

    row.index.vector <- 1:number.rows

    # Now we can perform the `cbind()` operation:

    indexed.data.frame <-
        cbind(
            row.index = row.index.vector,
            input.data.frame
        )

    # Finally, we want to return the indexed.data.frame as the output value:

    return( indexed.data.frame )
}
```

Let's see how the function `create.row.index()` works:

```
indexed.cars.data.frame.2 <-
    create.row.index( cars )

head( indexed.cars.data.frame.2 )
```

```
##   row.index speed dist
## 1         1     4    2
## 2         2     4   10
## 3         3     7    4
## 4         4     7   22
## 5         5     8   16
## 6         6     9   10
```

We can write the `create.row.index()` in a more compressed style:

```
create.row.index <- function( input.data.frame ) {

    return(
        cbind(
            row.index = 1:nrow( input.data.frame ),
            input.data.frame
        )
    )
}
```

This is more elegant than the previous version, but you don't have to do it this way, and if you feel more comfortable using the previous approach that's fine too.

Either way is acceptable, as long as it's readable and you can be sure that it's correct.

So that's how to bind columns to a data frame.

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

Module Review

In this module, we learned how to combine the rows or columns of data frames.

- In Section 1, we saw how to append rows to a data frame.
- In Section 2, we saw how to bind columns to a data frame.

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

- Combine two data frames together by binding rows, and explain the conditions that must be satisfied to perform this operation.
- Combine two data frames together by binding columns, and explain the conditions that must be satisfied to perform this operation.

There were 2 new built-in R functions in this module:

- `rbind()`
- `cbind()`

All right! That's it for Module 2: Binding Rows and Columns.

Now let's move on to Module 3: Merging Data Frames.