Week 10 Module 3: Merging Data Frames

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: Merging Data Frames.

In this module, we will explore the `merge()` function, which is an alternative method of combining columns of two data frames.

- In Section 1, we'll study the basic concept of merging.
- In Section 2, we'll learn how to performing a basic merge in R.
- In Section 3, we'll see how to modify the merge operation.
- In Section 4, we'll discuss some practical issues that arise when we merge data frames.

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

- Explain how the merging process differs from simply binding columns.
- Perform a simple merge in R.
- Modify the merge process in R to accomodate unmatched rows.

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

- `merge()`

All right! Let's get started by studying the basic concept of merging.

Section 1: Merging versus Binding Columns

> **Main Idea:** *We can combine the columns of two data frames by merging them.*

In this section, we'll study the basic concept of merging.

If we have two data frames and we want to combine their columns, we've seen that we can use the `cbind()` function.

However, this has a drawback – we have to be absolutely sure that the rows in the two data frames have to be exactly aligned.

If the rows don't align, then when we use the `cbind()` function the resulting data frame will have a corrupted row structure.

An alternative approach is to use what's called a "merge" operation.

Here, each row has some form of a unique identifier.

The unique identifier could be a single variable, or it could be a combination of variables.

However we structure the unique identifier, it must be the case that each row has its own identifier value.

If two data frames have a similar system of unique identifiers, then we can align rows by matching identifiers.

We all encounter this system of unique identifiers every day.

Every time you log into your email account, you have to supply a unique identifier, typically your email address.

That's basically what's going on here.

The system of identifiers allows us to overcome the problems with the `cbind()` function.

Now the rows *don't* have to be exactly aligned, because the merging process matches rows on the basis of unique identifiers.

However, the tradeoff is that it's your responsibility to set up and maintain the unique identifiers.

So that's the basic concept of merging two data frames together.

Now let's see how to implement this in R.

Section 2: Basic Merging

> **Main Idea:** *Let's use R to merge two data frames together*

In this section, we'll learn how to performing a basic merge in R.

Let's create two simple data frames to illustrate this process:

```r
merge.data.frame.1 <-
    data.frame(
        var.1 = c("a", "b", "c"),
        var.2 = c("x", "y", "z"),
        var.3 = c(1, 2, 3)
    )

merge.data.frame.1
```

```
##   var.1 var.2 var.3
## 1     a     x     1
## 2     b     y     2
## 3     c     z     3
```

```
merge.data.frame.2 <-
    data.frame(
        var.1 = c("a", "b", "c"),
        var.4 = c("p", "q", "r"),
        var.5 = c(4, 5, 6)
    )

merge.data.frame.2
```

```
##   var.1 var.4 var.5
## 1     a     p     4
## 2     b     q     5
## 3     c     r     6
```

We can merge these two data frames together by using the `merge()` function:

```
merge(
    merge.data.frame.1,
    merge.data.frame.2
)
```

```
##   var.1 var.2 var.3 var.4 var.5
## 1     a     x     1     p     4
## 2     b     y     2     q     5
## 3     c     z     3     r     6
```

Notice what happened here.

The `merge()` function combined the two data frames, and it matched the rows using the values in the column `var.1`.

Once the rows are matched, then R includes all the other variables to produce the combined row.

An interesting feature of R is that the data frames do not have to be sorted.

Many other software packages require that the user first sort the rows of the datasets before performing a merge, but R does not need this.

To see this, here's another data frame with the same values as `merge.data.frame.2`, but with the rows in a different order:

```
merge.data.frame.3 <-
    data.frame(
        var.1 = c("c", "b", "a"),
        var.4 = c("r", "q", "p"),
        var.5 = c(6, 5, 4)
    )

merge.data.frame.3
```

```
##   var.1 var.4 var.5
## 1     c     r     6
## 2     b     q     5
## 3     a     p     4
```

Now let's try the merge:

```
merge(
    merge.data.frame.1,
    merge.data.frame.3
)
```

```
##   var.1 var.2 var.3 var.4 var.5
## 1     a     x     1     p     4
## 2     b     y     2     q     5
## 3     c     z     3     r     6
```

How did `merge()` know that it was supposed to use the column `var.1` to perform the merge?

The `merge()` function scans the two data frames, and determines all the column names that are common to both `A` and `B`.

Then, by default, it uses the set of common column names as the basis for the merge matching.

This is a cool feature, but I think it's generally considered best practice to explicitly indicate the columns for the merging by using the `by` option, like this:

```
merge(
    merge.data.frame.1,
    merge.data.frame.2,
    by = c( "var.1" )
)
```

```
##   var.1 var.2 var.3 var.4 var.5
## 1     a     x     1     p     4
## 2     b     y     2     q     5
## 3     c     z     3     r     6
```

If you look in the documentation, you'll see some fancy code for the default value of `by`.

Essentially, R takes the intersection of the columns of the two data frames, and then uses these common column names for the merge matching.

So it's actually possible to use multiple columns to determine the unique identifier.

So that's how to perform a basic merge in R.

Now let's see how to modify the merge operation.

Section 3: Modifying the Merge Operation

Main Idea: *We can modify the merge operation*

In this section, we'll see how to modify the merge operation.

What happens if we want to merge two data frames together, but some of the rows don't match?

For instance, consider this data frame:

```
merge.data.frame.4 <-
    data.frame(
        var.1 = c("a", "b", "x"),
        var.4 = c("p", "q", "r"),
        var.5 = c(4, 5, 6)
    )

merge.data.frame.4
```

```
##   var.1 var.4 var.5
## 1     a     p     4
## 2     b     q     5
## 3     x     r     6
```

Now, when we try to merge the two data frames using the `var.1` variable, there will be a row in `merge.data.frame.1` that has a unique identifier that doesn't match anything in `merge.data.frame.4`, and vice versa.

What do we do about this?

First, let's see what the default behavior is in R:

```
merge(
    merge.data.frame.1,
    merge.data.frame.4
)
```

```
##   var.1 var.2 var.3 var.4 var.5
## 1     a     x     1     p     4
## 2     b     y     2     q     5
```

So the default behavior for R is to include only the rows that match between the two data frames.

Sometimes that's exactly what we want, but sometimes it isn't.

Notice that with this approach there was a row in `merge.data.frame.1` that doesn't get included in the final output data frame, because there wasn't a match in the `merge.data.frame.4` data frame.

Suppose we want to keep all the rows from `merge.data.frame.1`, and if there's a match with a row in `merge.data.frame.4`, that's great, but if there isn't a match we still want to keep the row.

We can control this by using the options `all.x` and `all.y`, which by default are set to `FALSE`.

If we set `all.x` to TRUE, then the `merge()` function will keep all the rows of the first input argument data frame, regardless of whether or not they match:

```
merge(
    merge.data.frame.1,
    merge.data.frame.4,
    by = "var.1",
    all.x = TRUE
)
```

```
##   var.1 var.2 var.3 var.4 var.5
## 1     a     x     1     p     4
## 2     b     y     2     q     5
## 3     c     z     3  <NA>    NA
```

Notice here that we keep the data for the row with a `var.1` value of `c`, even though there was no match with any rows in `merge.data.frame.4`.

The values of the variables `var.4` and `var.5` are `NA`, which makes sense, because there was no row in `merge.data.frame.4` that could supply this information.

If we want to keep all the rows from the second data frame but only include the rows from the first data frame that match, we can set `all.y` to `TRUE`:

```
merge(
    merge.data.frame.1,
    merge.data.frame.4,
    by = "var.1",
    all.y = TRUE
)
```

```
##   var.1 var.2 var.3 var.4 var.5
## 1     a     x     1     p     4
## 2     b     y     2     q     5
## 3     x  <NA>    NA     r     6
```

Again, notice the generation of `NA` values for unmatched rows.

If we want all the rows from both data frames, we can set both `all.x` and `all.y` to `TRUE`:

```
merge(
    merge.data.frame.1,
    merge.data.frame.4,
    by = "var.1",
    all.x = TRUE,
    all.y = TRUE
)
```

```
##   var.1 var.2 var.3 var.4 var.5
## 1     a     x     1     p     4
## 2     b     y     2     q     5
## 3     c     z     3  <NA>    NA
## 4     x  <NA>    NA     r     6
```

So that's how to modify the merge operation.

Now let's discuss some practical issues with merging.

Section 4: Practical Issues

Main Idea: *There are some issues that arise in practice when performing merge operations*

In this section, we'll discuss some practical issues that arise when we merge data frames.

Many projects involved multiple data sources, and these typically have to be merged at some point.

In principle, you only do the merge once, when you're assembling all the data into one combined data frame, and once you've done that, then you just work with the combined data frame.

So in theory you only do a few merges on a project, and once you've done them you're finished.

This depends on the nature of your project, and in some situations you might have to merge data frames together on a regular basis.

However, even there, once you've merged all your data frames together, then you typically don't do it again, at least for that analysis.

This is in contrast to statistical procedures, where you might run dozens of regression analyses in a single session.

In practice, merges can be difficult, because they are very finicky about the unique identifiers.

Subtle misspellings or coding inaccuracies with the unique identifiers can be very difficult to track down, especially if you're working with a large data frame.

You also have to make a number of potentially subtle design decisions: if you're doing a merge, and some of the rows don't match, what should happen?

There's no general answer, and the solution will always be specific to your project.

Checking the integrity of your merge will also be a project-specific undertaking.

For instance, if you have two data frames and you *know* that all the rows must match, then when you're done you can use the `nrow()` function to count the number of rows in the output merged data frame and make sure it's the right number.

You just have to have a strong understanding of how the `merge()` function operates, as well as the details of your data.

So those are some practical issues that arise when we merge data frames.

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

Module Review

In this module, we explored the `merge()` function, which is an alternative method of combining columns of two data frames.

- In Section 1, we studied the basic concept of merging.
- In Section 2, we learned how to performing a basic merge in R.
- In Section 3, we saw how to modify the merge operation.
- In Section 4, we discussed some practical issues that arise when we merge data frames.

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

- Explain how the merging process differs from simply binding columns.
- Perform a simple merge in R.
- Modify the merge process in R to accomodate unmatched rows.

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

- `merge()`

All right! That's it for Module 3: Merging Data Frames.

Now let's move on to Module 4: Stratified Charts.