Week 6 Module 1: Basic Concepts

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 1: Basic Concepts.

In this module, we'll explore the basic concepts of categorical data.

- In Section 1, we'll define the concept of categorical data.
- In Section 2, we'll explore the numeric representation of categorical data.
- In Section 3, we'll review the character string representation of categorical data.
- In Section 4, we'll meet factors, which combine the features of the numeric and character string representation methods.
- In Section 5, we'll compare and contrast factors with vectors.

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

- Define the concept of categorical data.
- Explain how to represent categorical data using numeric values.
- Explain how to represent categorical data using character string values.
- Explain the concept of a factor, and how it represents categorical data
- Construct factors from vectors.
- Compare and contrast properties of factors and vectors.

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

- `factor()`
- `is.factor()`

All right, let's get rolling by defining the basic concept of categorical data.

Section 1: Basic Concepts

> **Main Idea:* *Let's explore the basic concepts of categorical data*

In this section, we'll discuss the basic concepts of *categorical data*.

One type of data that arises frequently in practice is *categorical* data.

A categorical variable is a variable that can take on only a finite number of possible values.

These different values are effectively different categories.

When working with categorical data, we call the set of categories the "levels" of the categorical data.

For instance:

- A grocery store can sell 3 different brands of cereal, so there are 3 levels for the brands: Sugar Bomz, Krispee Yummm!!, and Healthy Kale and Tofu.
- In baseball, a hit must be one of four kinds, so there are 4 levels for hits: a single, a double, a triple, or a home run.
- A widget from WiDgT can be one of five different models, so there are 5 levels for these models: Classic WiDgT, WiDgT 2.0, WiDgT 3k, Quadcore WiDgT, and WiDgT Mach 5.

We usually think of categorical data in terms of a few categories, but there's no reason why you can't define a categorical variable with 10,000 different values.

In R, there are three different approaches to implementing categorical data values:

- The most basic is to use a numeric representation of the different category levels.
- Another approach is to use a character string representation of the levels.
- The third approach is to use a factor, which combines the features of the numeric and character string representation methods.

In the rest of this module, we'll explore these fundamental methods for representing categorical data.

So that's the basic concept of categorical data.

Now let's explore the numeric representation of categorical data.

Section 2: The Numeric Representation

Main Idea: *We can represent categorical data using numeric values*

In this section, we'll explore the numeric representation of categorical data.

How can we represent categorical data in R?

A common approach is to use numbers for each distinct category.

For instance, in the case of cereal brands, we could represent the three different brands with three different numeric values or "codes":

Brand	Numeric Code
Sugar Bomz	1
Krispee Yummm!!	2
Healthy Kale and Tofu	3

Let's suppose we have a sequence of cereal sales, and the brands for each transaction are:

Transaction	Brand
1	Sugar Bomz
2	Sugar Bomz
3	Healthy Kale and Tofu
4	Sugar Bomz
5	Krispee Yummm!!
6	Healthy Kale and Tofu

Then, using the method of numeric encoding, the brands for each transaction would be represented as:

Transaction	Brand	Numeric Representation
1	Sugar Bomz	1
2	Sugar Bomz	1
3	Healthy Kale and Tofu	3
4	Sugar Bomz	1
5	Krispee Yummm!!	2
6	Healthy Kale and Tofu	3

And that's what you see in the dataset – just the sequence of numeric codes, and **not** any character string representation.

So, to do this in R we would have:

```r
transaction.brand.numeric.vector <-
  c( 1, 1, 3, 1, 2, 3 )
```

This isn't wrong, but it's not the best approach.

The main problem with using a numerical representation is that there's no obvious way to recover the actual brand names from the numerical value.

The solution in practice is to construct a "key", which is a document that indicates what each numerical value represents.

But now there is a separate document that has to be included with the dataset, and it's easy to lose this document or to forget to bundle it with the dataset.

If categorical data is encoded using a numerical representation and the key is lost, then the data is lost as well.

Another problem with using a numerical representation is that it can be tedious to enter this sort of data, and it's very difficult to review it to check for mistakes.

As I said, there's nothing *wrong* with using a numerical representation, strictly speaking, and if you do it correctly you'll always get the right answer.

But I think it can also lead to some very unpleasant problems, and I don't recommend it as a workflow best practice.

Having said this, I must admit that many people find this method attractive, because they understand how to work with numbers and they don't understand the techniques that we will explore in this lecture.

If you are in a group where the standard convention is to use this form of numeric representation for the values of the categorical variable, you *must* be highly disciplined about maintaining the proper documentation.

So that's the method of representing categorical data using numeric values.

Now let's see how to represent categorical data using character string values.

Section 3: The Character String Representation of Categorical Data

Main Idea: *We can represent categorical data using character string values*

In this section, we'll review the character string representation of categorical data.

Let's return to the sequence of cereal sales, and the brands for each transaction are:

Transaction	Brand
1	Sugar Bomz
2	Sugar Bomz
3	Healthy Kale and Tofu
4	Sugar Bomz
5	Krispee Yummm!!
6	Healthy Kale and Tofu

Now we're going to represent each cereal brand with a short character string identifier or "label":

Brand	Character String Label
Sugar Bomz	"SBZ"
Krispee Yummm!!	"KYM"
Healthy Kale and Tofu	"HKT"

Now we have:

Transaction	Brand	Character String Label
1	Sugar Bomz	"SBZ"
2	Sugar Bomz	"SBZ"
3	Healthy Kale and Tofu	"HKT"
4	Sugar Bomz	"SBZ"
5	Krispee Yummm!!	"KYM"
6	Healthy Kale and Tofu	"HKT"

And that's what you see in the dataset – just the sequence of character string labels.

So, to do this in R we would have:

```r
cereal.brand.character.string.vector <-
  c( "SBZ", "SBZ", "HKT", "SBZ", "KYM", "HKT" )
```

You might be wondering how this is in any way different from what we've been doing all along in this course.

It's not.

In fact, we've implicitly been using the character string representation for categorical data without really discussing it in detail, so you can see how intuitive this method is.

That's why I used the word "review" at the beginning of this section.

The character string representation avoids the two major problems with the numeric representation.

First, we don't need to maintain a separate "key", and the data is essentially self-documenting if we use clear, easy-to-understand character strings.

Second, it's much easier to input and review data when working with words rather than numbers.

In practice, I think many people avoid this approach because they don't understand how to work with character string vectors, and one of the primary goals of this course is to help you develop your skills so that you can be comfortable with character string vectors.

So that's how to represent categorical data using character string values.

Now let's meet factors, which combine the features of the numeric and character string representation methods.

Section 4: Factors

> **Main Idea:** *We can use factors to represent categorical data*

In this section, we'll meet factors, which combine the features of the numeric and character string representation methods.

R uses a special data structure called a *factor* to represent categorical data.

In my opinion, one of the signs that someone genuinely understands how to program in R is that they can correctly work with factors.

Many people cannot do this, and so this lecture is particularly important.

Factors combine the numerical and character string approaches to represent categorical values.

A factor represents a categorical variable by using two structures:

- Internally, the categorical data is represented as an integer, so this is a form of numeric representation.
- The numeric codes are associated with character string labels, so this is a form of character string representation.

By using a numeric vector as the underlying implementation, factors obtain the speed and efficiency gains of a numeric representation.

By associating the numeric codes with character string identifiers, factors make it easier for us to work with and think about the data.

In essence, a factor is a numeric representation of categorical data where we've actually stored the key in the data structure itself.

Let’s return to the sequence of cereal sales, and the brands for each transaction are:

Transaction	Brand
1	Sugar Bomz
2	Sugar Bomz
3	Healthy Kale and Tofu
4	Sugar Bomz
5	Krispee Yummm!!
6	Healthy Kale and Tofu

Now we’re going to set up a table associating each different brand with a numeric code:

Brand Name	Numeric Code	Level
Sugar Bomz	1	“SBZ”
Krispee Yummm!!	2	“KYM”
Healthy Kale and Tofu	3	“HKT”

Then we’ve seen that the numeric representation for this data is:

Transaction	Brand	Numeric Representation
1	Sugar Bomz	1
2	Sugar Bomz	1
3	Healthy Kale and Tofu	3
4	Sugar Bomz	1
5	Krispee Yummm!!	2
6	Healthy Kale and Tofu	3

The factor also includes the character string label as well:

Transaction	Brand	Numeric	Character String Label
1	Sugar Bomz	1	“SBZ”
2	Sugar Bomz	1	“SBZ”
3	Healthy Kale and Tofu	3	“HKT”
4	Sugar Bomz	1	“SBZ”
5	Krispee Yummm!!	2	“KYM”
6	Healthy Kale and Tofu	3	“HKT”

Let’s see how to do this in R.

To construct a factor, we use the `factor()` function, which takes a vector as its input argument:

```
cereal.brand.character.string.vector
```

```
## [1] "SBZ" "SBZ" "HKT" "SBZ" "KYM" "HKT"
```

```
cereal.brand.factor <-
  factor(
    x = cereal.brand.character.string.vector
  )

cereal.brand.factor
```

```
## [1] SBZ SBZ HKT SBZ KYM HKT
## Levels: HKT KYM SBZ
```

Notice that the factor displays a little differently from how the character string vector displays:

```
cereal.brand.character.string.vector
```

```
## [1] "SBZ" "SBZ" "HKT" "SBZ" "KYM" "HKT"
```

For one thing, the character string vector displays each value enclosed in quotes, while the factor does not use quotes.

Also, the factor explicitly reports the different levels.

We can use the `as.character()` function to recover the vector of character string labels:

```
as.character( cereal.brand.factor )
```

```
## [1] "SBZ" "SBZ" "HKT" "SBZ" "KYM" "HKT"
```

We can use the `as.numeric()` function to recover the vector of the numeric codes:

```
as.numeric( cereal.brand.factor )
```

```
## [1] 3 3 1 3 2 1
```

So that's how factors can represent categorical data.

Now let's compare factors with vectors.

Section 5: Factors vs. Vectors

Main Idea: *Let's compare factors and vectors*

In this section, we'll compare and contrast factors with vectors.

When we display a factor, it looks very much like a vector.

Here for review is our original character vector:

```
cereal.brand.character.string.vector
```

```
## [1] "SBZ" "SBZ" "HKT" "SBZ" "KYM" "HKT"
```

And here is the factor version:

```
cereal.brand.factor
```

```
## [1] SBZ SBZ HKT SBZ KYM HKT
## Levels: HKT KYM SBZ
```

You can see that the vector and the factor have many features in common, but they also differ in their display.

Naturally enough, the simple character vector is a member of the `character` class:

```
class( cereal.brand.character.string.vector )
```

```
## [1] "character"
```

However, the factor version is not of the `character` class, but instead is a different class, named appropriately enough `factor`:

```
class( cereal.brand.factor )
```

```
## [1] "factor"
```

Notice that the factor is not a vector:

```
is.vector( cereal.brand.factor )
```

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

Likewise, the vector is not a factor:

```
is.factor( cereal.brand.character.string.vector )
```

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

So the factor and the vector really are different kinds of objects.

However, we can separate a factor into its components, and these will be vectors.

To obtain the levels of a factor we can use the `levels()` function, and this will return

```
levels( cereal.brand.factor )
```

```
## [1] "HKT" "KYM" "SBZ"
```

If we coerce the original factor to a character class, we obtain a vector of the character string labels:

```
as.character( cereal.brand.factor )
```

```
## [1] "SBZ" "SBZ" "HKT" "SBZ" "KYM" "HKT"
```

This is just the original character vector.

Likewise, we can obtain the vector of numeric representatives by coercing the factor to the `numeric` class:

```r
as.numeric( cereal.brand.factor )
```

```
## [1] 3 3 1 3 2 1
```

In some respects, vectors and factors are very similar.

For instance, both are one-dimensional structures, and the `length()` function returns the number of elements in either factors or vectors.

Let's count the number of elements in the character string vector:

```r
length( cereal.brand.character.string.vector )
```

```
## [1] 6
```

We can count the number of elements in the factor in the exact same way:

```r
length( cereal.brand.factor )
```

```
## [1] 6
```

Also, you can use positive integer indexing with factors:

```r
cereal.brand.factor[ 2 ]
```

```
## [1] SBZ
## Levels: HKT KYM SBZ
```

Notice that what you get back is a factor.

You can also do logical indexing on the factor:

```r
cereal.brand.factor[ c( TRUE, TRUE, FALSE, TRUE, FALSE, TRUE ) ]
```

```
## [1] SBZ SBZ SBZ HKT
## Levels: HKT KYM SBZ
```

We can also construct logical vectors by using a vectorized comparison operation on a factor:

```r
cereal.brand.factor == "SBZ"
```

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

So for instance we can count the number of transactions for Sugar Bomz by using logical indexing and the `sum()` function:

```r
sum( cereal.brand.factor == "SBZ" )
```

```
## [1] 3
```

In these ways, the factor very much "feels" like a character vector.

Factors are very closely related to named vectors.

Sometimes it can be hard to tell the difference, especially if the named vector is a numeric vector.

After all, in both cases, we have a numeric vector, and a vector of character strings that is somehow associated with the numeric vector.

In fact, named vectors and factors are complementary, and are quite different from one another.

With a named vector, we use character strings to label the locations of the vector.

Let's recall out pricing vector for the cereal brands:

```
price.vector <-
    c(
        sbz = 2.99,
        kym = 3.49,
        hkt = 7.99
    )

price.vector
```

```
##  sbz  kym  hkt
## 2.99 3.49 7.99
```

Notice that each location has a distinct character string associated with it.

The number of character string names for a named vector has to be the same as the number of elements of the vector.

For instance, if the vector has length 37, then we need 37 different character strings to label all the locations uniquely.

Since the character strings all have to be different, the names of the locations serve as unique identifiers for the locations.

The role of the character strings with factors is completely different.

With a factor, the character strings are labels for the different *values* of the categorical variable.

Thus, the `cereal.brand.factor` has 6 locations, but only 3 levels, because there are only 3 different values for this categorical variable.

```
length( cereal.brand.factor )
```

```
## [1] 6
```

We only have three different levels:

```
length( levels( cereal.brand.factor ) )
```

```
## [1] 3
```

Let's summarize the fundamental difference between named vectors and factors:

- With named vectors, the character strings are identifiers for the different locations of the vector.

- With factors, the character strings are identifiers for the different values of the variable.

So that's how vectors and factors compare.

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

Module Review

In this module, we explored the basic concepts of categorical data.

- In Section 1, we defined the concept of categorical data.
- In Section 2, we explored the numeric representation of categorical data.
- In Section 3, we reviewed the character string representation of categorical data.
- In Section 4, we met factors, which combine the features of the numeric and character string representation methods.
- In Section 5, we compared and contrasted factors with vectors.

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

- Define the concept of categorical data.
- Explain how to represent categorical data using numeric values.
- Explain how to represent categorical data using character string values.
- Explain the concept of a factor, and how it represents categorical data.
- Construct factors from vectors.
- Compare and contrast properties of factors and vectors.

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

- `factor()`
- `is.factor()`

All right, that's it for Module 1: Basic Concepts.

Now let's move on to Module 2: Factors.