Week 6 Module 4: Barplots

CSCI 5a: Programming in R

Let's clear the global environment:

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

Now let's load in the Module 4 R objects:

```
load( "Module 4 R Objects.Rdata" )
```

Module Overview and Learning Objectives

Hello! And welcome to Module 4: Barplots.

In this module, we'll learn how to construct *barplots*.

- In Section 1, we'll learn about the basic concepts of barplots.
- In Section 2, we'll see how tables can help us to construct barplots.
- In Section 3, we'll see how to use factors to control aspects of the barplot display.
- In Section 4, we'll learn how to display bars for unobserved factor levels.

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

- Explain the basic concept of a barplot.
- Construct a basic barplot using R.
- Use the `table()` function to prepare data for the barplot.
- Use factors to control aspects of the barplot display, including bars for unobserved values.

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

- `barplot()`

Section 1: Barplot Basics

Main Idea: *Let's construct a barplot*

In this section, we'll learn about the basic concepts of barplots.

Barplots are among the most powerful and expressive data visualization methods.

However, there is a learning curve for all this flexibility, and we need to spend some time learning special techniques.

Suppose we are working with a categorical variable such that for each level there is an associated numeric value.

For instance, a grocery store might be interested in the number of boxes of cereal sold, for three cereal brands.

Then the cereal brand is the level, and the associated numeric value is the number of boxes sold:

Brand	Number of Boxes Sold
Healthy Kale and Tofu	17
Sugar Bomz	129
Krispy Yummm	98

A *barplot* displays this information by drawing a bar for each category.

The height of the bar indicates the associated numeric value.

Let's store these numeric values in a vector:

```
boxes.sold.frequency.count.vector <-
    c(17, 129, 98)
```

Let's make a barplot of the associated numeric values in the vector:

```
barplot( height = boxes.sold.frequency.count.vector )
```

Make sure before you move on that you understand how the height of each bar is determined.

Our barplot isn't wrong, but it's not as useful as it could be.

For one thing, it doesn't display the names of the categories.

We can label the bars using the `names.arg` option:

```
barplot(
    height = boxes.sold.frequency.count.vector,
    names.arg =
        c( "HKT", "SBZ", "KYM" )
)
```

Of course, it would be nice to have a main title and titles for the x- and y-axes:

```
barplot(
    height = boxes.sold.frequency.count.vector,
    names.arg =
        c( "HKT", "SBZ", "KYM" ),
    main = "Barplot of cereal sales",
    xlab = "Cereal brand",
    ylab = "Number of boxes sold"
)
```

We can adjust the range of the y-values using the `ylim` option:

```
barplot(
    height = boxes.sold.frequency.count.vector,
    ylim = c(0, 200),
    names.arg =
        c( "HKT", "SBZ", "KYM" ),
    main = "Barplot of cereal sales",
    xlab = "Cereal brand",
    ylab = "Number of boxes sold"
)
```

Finally, it would be nice to adjust the colors, and we can do this with the `col` parameter:

```
barplot(
    height = boxes.sold.frequency.count.vector,
    names.arg =
        c( "HKT", "SBZ", "KYM" ),
    main = "Barplot of cereal sales",
    xlab = "Cereal brand",
    ylab = "Number of boxes sold",
    ylim = c(0, 150),
    col = "cadetblue1",
    las = 1
)
```

When R makes a barplot, it doesn't "know" what the height of the bars is supposed to mean.

It's up to us to come up with the right numbers.

R just takes what we give it and draws the plot.

This means that the `barplot()` function has great flexibility, and we can use it to visualize just about any numeric values associated with a category.

For instance, instead of making a barplot of the absolute counts, let's make a barplot of the relative proportions.

First, we'll construct a vector consisting of the relative proportions:

```
total.number.of.boxes.sold <-
    sum( boxes.sold.frequency.count.vector )

boxes.sold.proportion.vector <-
    boxes.sold.frequency.count.vector /
    total.number.of.boxes.sold
```

```
boxes.sold.proportion.vector
```

Now we can make the barplot using the relative proportions:

```
barplot(
    height = boxes.sold.proportion.vector,
    names.arg =
        c( "HKT", "SBZ", "KYM" ),
    main = "Barplot of cereal sales",
    xlab = "Cereal brand",
    ylab = "Relative proportion of boxes sold",
    ylim = c(0, 150),
    col = "cadetblue"
)
```

Whoops! We'd better adjust the range of the y-axis:

Now we can make the barplot using the relative proportions:

```
barplot(
    height = boxes.sold.proportion.vector,
    names.arg =
        c( "HKT", "SBZ", "KYM" ),
    main = "Barplot of cereal sales",
    xlab = "Cereal brand",
    ylab = "Relative proportion of boxes sold",
    ylim = c(0, 0.6),
    col = "cadetblue"
)
```

So that's the basic concept of a barplot.

Now let's see how to use tables to construct barplots.

Exercise 1: Simple barplot

Here are the number of support calls for each of the regional offices of WiDgT:

Location	Number of support calls
Boston	43
London	27
Salt Lake City	11
Shanghai	56

Construct a barplot that displays the frequency counts of the number of support calls across the locations.

Solution

Section 2: Barplots and Tables

Main Idea: *we can use tables to construct barplots*

In this section, we'll see how to use tables to help us to construct barplots.

We'll often want to draw a barplot using observed data.

Suppose we have this sequence of transactions:

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

We can tabulate the sample frequency counts for each brand:

Cereal Brand	Frequency Count
Healthy Kale and Tofu	1
Krispee Yummm!!	4
Sugar Bomz	3

We could then use this to construct a barplot of the frequency counts:

```
barplot(
    height = c(1, 4, 3),
    names.arg =
        c( "HKT", "KYM", "SBZ" ),
    main = "Barplot of cereal sales",
    xlab = "Cereal brand",
    ylab = "Number of boxes sold",
    ylim = c(0, 5),
    col = "cadetblue"
)
```

R can perform this tabulation operation automatically using the `table()` function.

Let's store the sequence of observed values in a vector:

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

Now we can use the `table()` function to construct a table of these observed values:

```
cereal.sales.frequency.count.table <-
    table( cereal.sales.character.string.vector )

cereal.sales.frequency.count.table
```

Now we can make a barplot using this table:

```
barplot(
    height = cereal.sales.frequency.count.table,
    ylim = c(0, 5),
    main = "Barplot of observed counts",
    xlab = "Cereal brand",
    ylab = "Number of boxes sold",
    col = c("chocolate2" )
)
```

Notice that the `table()` function orders the categories alphabetically, because it's basically working with a factor, and that's the default method for constructing a factor.

What happens if we want to display the bars in a different order?

One way to achieve this is to use the `sort()` function on the table:

```
sorted.cereal.sales.frequency.count.table <-
    sort( cereal.sales.frequency.count.table )

sorted.cereal.sales.frequency.count.table
```

We can then construct the barplot with this ordering:

```
barplot(
    height = sorted.cereal.sales.frequency.count.table,
    ylim = c(0, 5),
    main = "Barplot of observed counts",
    xlab = "Cereal brand",
    ylab = "Number of boxes sold",
    col = c("chocolate1" )
)
```

By default, the `sort()` function sorts values by increasing order.

If we want to sort in decreasing order, we can use the `decreasing` option:

```
decreasing.cereal.sales.frequency.count.table <-
    sort(
      cereal.sales.frequency.count.table,
      decreasing = TRUE
    )
```

Now we can make the barplot with the bars in decreasing order:

```
barplot(
    height = decreasing.cereal.sales.frequency.count.table,
    ylim = c(0, 5),
    main = "Barplot of observed counts",
    xlab = "Cereal brand",
    ylab = "Number of boxes sold",
    col = c("chocolate2" )
)
```

So that's how to use tables to construct barplots.

Now let's see how to use factors to control barplots.

Exercise 2: Barplots and tables

Let's return to the `one.week.cereal.brand.character.string.vector` from the last module:

```
head( one.week.cereal.brand.character.string.vector )
```

First, construct a table of the frequency counts.

Then create a properly formatted barplot for this table.

Finally, construct a table of relative frequencies for this data, and make another properly formatted barplot.

Solution

Section 3: Barplots and Factors

> **Main Idea:** *we can use factors to control the display of a barplot*

In this section, we'll see how to use factors to control aspects of the barplot display.

There's another more general method for controlling the order of the bars, and this is where factors become extremely useful.

When we pass a vector to the `table()` function, the first thing that happens is that the vector is converted to a factor.

Remember that the default method here is for R to sort the levels of the factor alphabetically.

The `table()` function will use the ordering of the levels of the factor, so if we want to re-order how the categories are presented in the table (and therefore the barplot) we need to make that adjustment in the factor *before* we construct the table.

For instance, suppose we want the barplot to first display the bar for Sugar Bomz, then Healthy Kale and Tofu, and finally Krispee Yummm!!.

We won't be able to do this using the `sort()` function.

However, we can achieve it using factors.

We'll start by creating a factor from our cereal sales vector.

If we specify the levels, then R will use that ordering:

```
cereal.sales.factor <-
    factor(
        cereal.sales.character.string.vector,
        levels = c("SBZ", "HKT", "KYM" )
    )

cereal.sales.factor
```

Notice that now the categories are *not* listed in strictly alphabetical order, but instead they are listed in the order that we specified when we created the factor.

Now we can make the table:

```
reordered.cereal.sales.table <-
    table( cereal.sales.factor )

reordered.cereal.sales.table
```

Now let's make our barplot using this table:

```
barplot(
    height = reordered.cereal.sales.table,
    ylim = c(0, 5),
    main = "Barplot of observed counts",
    xlab = "Cereal brand",
    ylab = "Number of boxes sold",
    col = c("chocolate4" )
)
```

So that's how to use factors to control the barplot display.

Now let's see another application of factors to barplots.

Exercise 3: Barplots and factors

In the previous exercise, we created a barplot of the frequency counts for the one-week cereal sales.

However, the order of the categories wasn't our usual one, and it would be nice to display the data with our standard conventions.

First, construct a factor where the levels are ordered with Sugar Bomz first, then Krispee Yummm!!, and finally Healthy Kale and Tofu. Also, change the labels of the levels so that they are "SBZ", "KYM", and "HKT", respectively.

Next, construct a frequency count table using this factor, and save this in a variable.

Then create a properly formatted barplot for this table.

Solution

Section 4: Factors and Unrealized Levels

Main Idea: *We can use factors to display unrealized levels*

In this section, we'll learn how to display bars for unobserved factor levels.

Let's suppose we observe this sequence of cereal sales:

Transaction	Cereal Purchased
1	SBZ
2	KYM
3	SBZ
4	SBZ
5	SBZ
6	KYM
7	SBZ
8	KYM

Let's put this in a vector:

```
cereal.sales.vector <-
    c( "SBZ", "KYM", "SBZ", "SBZ",
       "SBZ", "KYM", "SBZ", "KYM")
```

What happens if we construct a table using just the observed values?

```
cereal.sales.table <-
    table( cereal.sales.vector )

cereal.sales.table
```

Now if we use this table to make a barplot we have:

```
barplot(
    height = cereal.sales.table,
    ylim = c(0, 5),
    main = "Barplot of observed counts",
    xlab = "Cereal brand",
    ylab = "Number of boxes sold",
    col = c("azure3" )
)
```

This isn't technically wrong, and the bars accurately reflect the observed data.

But it would be nice to have a bar for the "HKT" category.

We can say that the Healthy Kale and Tofu level was "unrealized", since we never actually observed an element with this value.

Just because there were no actual sales of Healthy Kale and Tofu in our limited sample doesn't mean that it shouldn't show up in the graph.

The value 0 is a numeric outcome, and we would like to see that "HKT" was observed 0 times.

The problem is that when R constructs the table from the observed values, it doesn't know about other values that weren't observed.

We can fix this by specifying the factor levels.

```
cereal.sales.factor <-
    factor(
        cereal.sales.vector,
        levels = c("SBZ", "KYM", "HKT")
    )
```

Now let's make the table using this factor:

```
cereal.sales.table <-
    table( cereal.sales.factor )

cereal.sales.table
```

Now we can make the barplot:

```
barplot(
    height = cereal.sales.table,
    ylim = c(0, 5),
    main = "Barplot of observed counts",
    xlab = "Cereal brand",
    ylab = "Number of boxes sold",
    col = c("azure1" )
)
```

So that's how to use factors to display unrealized levels.

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

Module Review

In this module, we learned how to construct *barplots*.

- In Section 1, we learned about the basic concepts of barplots.
- In Section 2, we saw how tables can help us to construct barplots.
- In Section 3, we saw how to use factors to control aspects of the barplot display.
- In Section 4, we learned how to display bars for unobserved factor levels.

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

- Explain the basic concept of a barplot.
- Construct a basic barplot using R.
- Use the `table()` function to prepare data for the barplot.
- Use factors to control aspects of the barplot display, including bars for unobserved values.

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

- `barplot()`

Allright, that's the end of Unit 7: Categorical Data

Now let's move on to Unit 8: Functions.

See you there!

Solutions to the Exercises

Exercise 1: Simple barplot

Here are the number of support calls for each of the regional offices of WiDgT:

Location	Number of support calls
Boston	43
London	27
Salt Lake City	11
Shanghai	56

Construct a barplot that displays the frequency counts of the number of support calls across the locations.

Solution

```
support.calls.vector <-
  c( 43, 27, 11, 56 )

barplot(
  height = support.calls.vector,
  ylim = c(0, 60),
  main = "Support calls across regional offices",
  xlab = "Locations",
  ylab = "Frequency count",
  col = "lightsteelblue2",
  names.arg = c( "Bos", "Lon", "SLC", "Sha" ),
)
```

Exercise 2: Barplots and tables

Let's return to the `one.week.cereal.brand.character.string.vector` from the last module:

```
head( one.week.cereal.brand.character.string.vector )
```

First, construct a table of the frequency counts for this data.

Then create a properly formatted barplot for this table.

Solution

```
cereal.brand.frequency.count.table <-
  table( one.week.cereal.brand.character.string.vector )

barplot(
  height = cereal.brand.frequency.count.table,
  main = "One-Week Cereal Brand Sales",
  xlab = "Cereal brand",
  ylab = "Frequency count",
  col = "azure2"
)
```

Exercise 3: Barplots and factors

In the previous exercise, we created a barplot of the frequency counts for the one-week cereal sales.

However, the order of the categories wasn't our usual one, and it would be nice to display the data with our standard conventions.

First, construct a factor where the levels are ordered with Sugar Bomz first, then Krispee Yummm!!, and finally Healthy Kale and Tofu. Also, change the labels of the levels so that they are "SBZ", "KYM", and "HKT", respectively.

Next, construct a frequency count table using this factor, and save this in a variable.

Then create a properly formatted barplot for this table.

Solution

First, we'll construct the factor:

```
one.week.cereal.brand.factor <-
  factor(
    x = one.week.cereal.brand.character.string.vector,
    levels = c( "Sugar Bomz", "Krispee Yummm!!", "Healthy Kale and Tofu" ),
    labels = c( "SBZ", "KYM", "HKT" )
  )
```

Next, we'll create the frequency count table, and save this in a variable:

```
one.week.cereal.brand.frequency.count.table <-
  table( one.week.cereal.brand.factor )

one.week.cereal.brand.frequency.count.table
```

Now we can create the barplot:

```
barplot(
  height = one.week.cereal.brand.frequency.count.table,
  ylim = c(0, 200),
  main = "Barplot of one-week sales across cereal brands",
  xlab = "Cereal brands",
  ylab = "Number of boxes sold",
  col = "darkseagreen2"
)
```