Unofficial Kahn Academy R Supplement: Categorical Data Displays

This is an R notebook/series of videos that I am writing to help some family members learn basic statistics and R coding simultaneously, so that statistics can be learned using practical, real-life analytical exercises. This way, people can learn the concepts while also learning how to execute them in a real-life data analysis situation. Traditionally statistics is taught first, then after a few years students develop coding skills. There really is no reason for this staggered approach, and it hinders students from developing their skills as hands-on analysts.

These instructionals are meant to accompany the Kahn Academy (hereafter KA) lectures on YouTube. However, I am not associated with Kahn Academy in any capacity.

R is quickly becoming the primary statistical software for most fields. Additionally, it is both free and cutting-edge in terms of its statistical capabilities. There are simpler programs out there, but they usually cost money (a lot of money, more than most Kahn Academy students learning stats can afford), aren't as flexible (you're stuck doing what the program wants you to do), and are maybe 10-20 years behind R in terms of what statistical methods they use, since cutting-edge techniques are often first programmed in R.

The notebook portion of this is available at my GitHub page, and I will be posting these videos online on YouTube.

R is an "object oriented program", which is a fancy way of saying that, whatever you are working with, whether it's a string of numbers, a spreadsheet, a formula, or a table of statistics, R treats it as an "object" that is stored over here on the right. Like many programs you can create objects that are assigned numbers, so for example, a<-2 (or, perhaps more intuitively a=2, in R they mean the same thing), creates and object **a** with a value 2 (here **objects** are in bold, commands are in *italics*).

This object **a** can sit up in your workspace along with a dataset (a table of information like a spreadsheet), and whatever else you need. This is nice when you are working with multiple datasets. R is available as a very bare-bones program where you type straight code into a box, but most people use an additional program layered on top of R called RStudio (which we are using here), which provides some additional helpful tools and makes R much easier to handle.

The first Kahn Academy section we'll start with is on categorical data displays. If you go through these movies, you should understand all the statistics you need to know for this supplementary material. Now all you need to learn is how to code.

https://www.khanacademy.org/math/statistics-probability/displaying-describing-data/categorical-data-displays

To create a bar plot we are going to use the famous "mtcars" dataset. This is a built-in dataset, which means that it comes with R and there is no need to load it. If you want to use your own dataset, you will need to load it and it will appear in the workspace. There are dozens of videos and other online sources showing you "how to load data into R" (look it up on Google), so we won't go into that here (with one exception since it caused me a lot of consternation in my early R days–when referring to a place on your computer where you are storing your files, make back-slashes [\] forwarded flashes [/]).

Since the mtcars dataset is built-in, it doe not naturally appear in the workspace, so I'm going to make an mtcars object by assigning the mtcars dataset to an object labeled **mtcars_data** (objects can't have spaces, so all object names have to be one word; traditionally object names with multiple words use underscores to separate the words).

```
mtcars_data<-mtcars
```

You can now click on the **mtcars_data** object and view it like you would a spreadsheet.

As you can see (hereafter AYCS), it has 11 characteristics of 32 types of cars. (You can also know this from the brief descriptor in the workspace.)

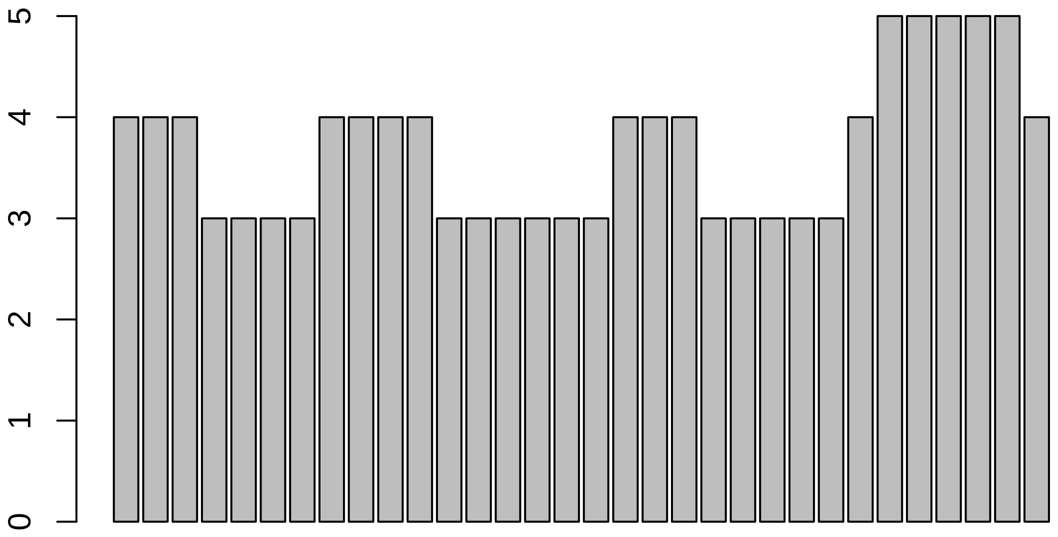
Because R can hold multiple datasets at a time, when you refer to a variable you also have to refer to the dataset that the variable is from. To do this you type the name of the dataset, then the $ sign, then the name of the variable.

Anyway, back to KA; the lesson above starts with how to read a bar plot. In R bar plots are easily made with the *barplot* command (who figured–you'll quickly learn that R does not make things more complicated than they need to be).

In R commands use parentheses to enclose the stuff you want the command to work on. In this case, we want to use *barplot* to create a bar plot of a variable in **mtcars**.

We will do a bar plot of how cars and gear number (using the variable **gear** in the **mtcars_dataset**:

```
barplot(mtcars_data$gear)
```



```
#Note: In some cases you may get an "Error in plot.new() : figure margins too large" message.
#If so, you can reset your margins using this command (details aren't important at this point)

par(mar=c(1,1,1,1))

#If you get an error message and you're not quite clear what it means in terms of
#what you're doing wrong,you can Google the erorr message and find a Stack Exchange
#or other post that explains what's wrong.
```

AYCS, this is not a very interesting graph. It shows 32 bars (one for each car) in sequential order showing how many gears there are. Bar plots are more often used to see how many of an item there is per value, so in this case how many cars have four gears, how many have three gears, etc.

To do this you have to make another dataset from the original dataset, in this case a dataset where the row is the gear number and the column is the number of cars that have that number of gears. This table is called a "frequency table," and is discussed in more detail in the Kahn Academy unit on "dot plots and frequency tables": https://www.khanacademy.org/math/statistics-probability/displaying-describing-data#dot-plots-frequency-tables

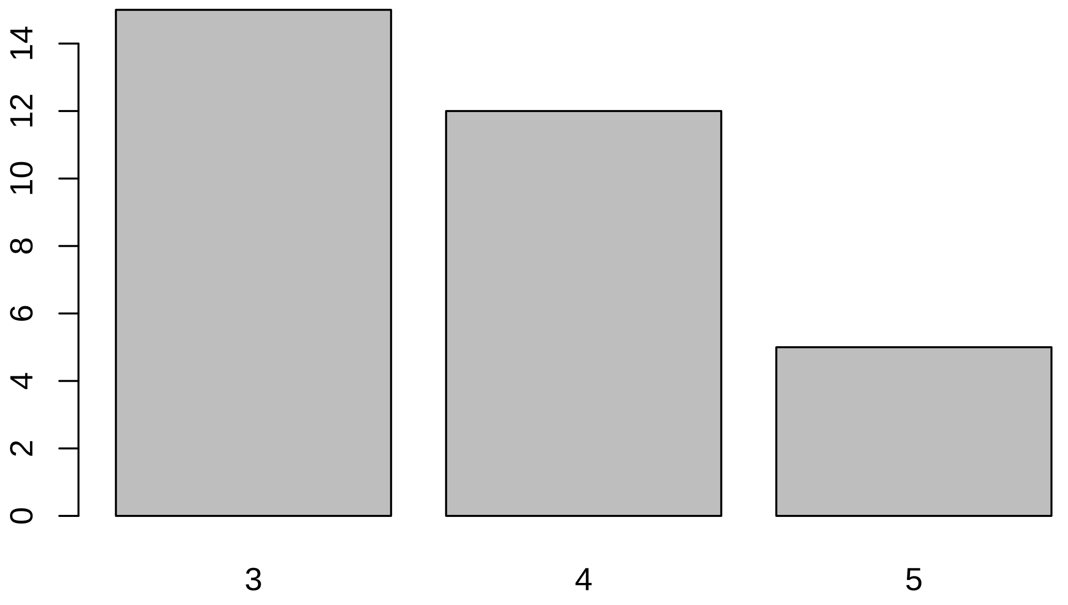
You can easily make a frequency table using the *table* command. The *table* command is useful if you want to quickly look at how many cases fit in each category/level of a variable in your data. For example, if you have a spreadsheet showing survey respondents and their marital status, income, education, etc,. and you wanted to look specifically at marital status to see how many are married, how many are divorced, how many have never been married, etc. then you can simply use the *table* command. In this case, we can see how many cars have each number of gear.

```
table(mtcars_data$gear)
```

```
##
##  3  4  5
## 15 12  5
```

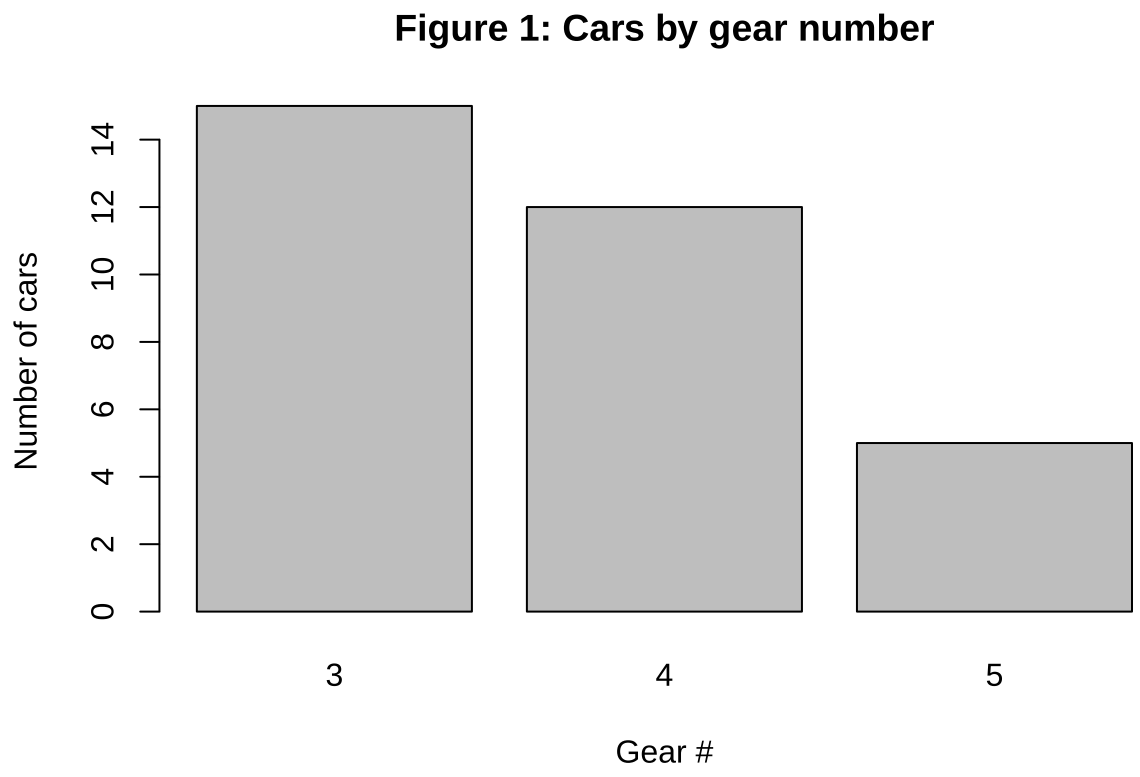
AYCS, the table command shows that there are 15 cars with three gears, 12 cars with four gears, and five cars with five gears. We can turn this table into its own object simply by assigning it a name (here, gear_table). Once this table/object is created, we can do the bar plot on it instead of the whole data.

```
gear_table<-table(mtcars_data$gear)
barplot(gear_table)
```



Different commands in R can spruce up this graph visually. In R, when you want to do additional, supplementary things like add a title to your graph and such, usually there is a sub-command that comes after the initial command but still within the parentheses. These sub-commands allow you to customize what you are doing. Each command has its own sub-commands, but many work across different commands. To get a sense of what customizations (sub-commands) you can do, look up the R documentation for that command (if you Google the name of that command followed by “r command”, it’s usually one of the top hits; also, you can often find examples of code using that command, which can be very helpful). In this case, adding a title and a label on our X-axis is relatively easy:

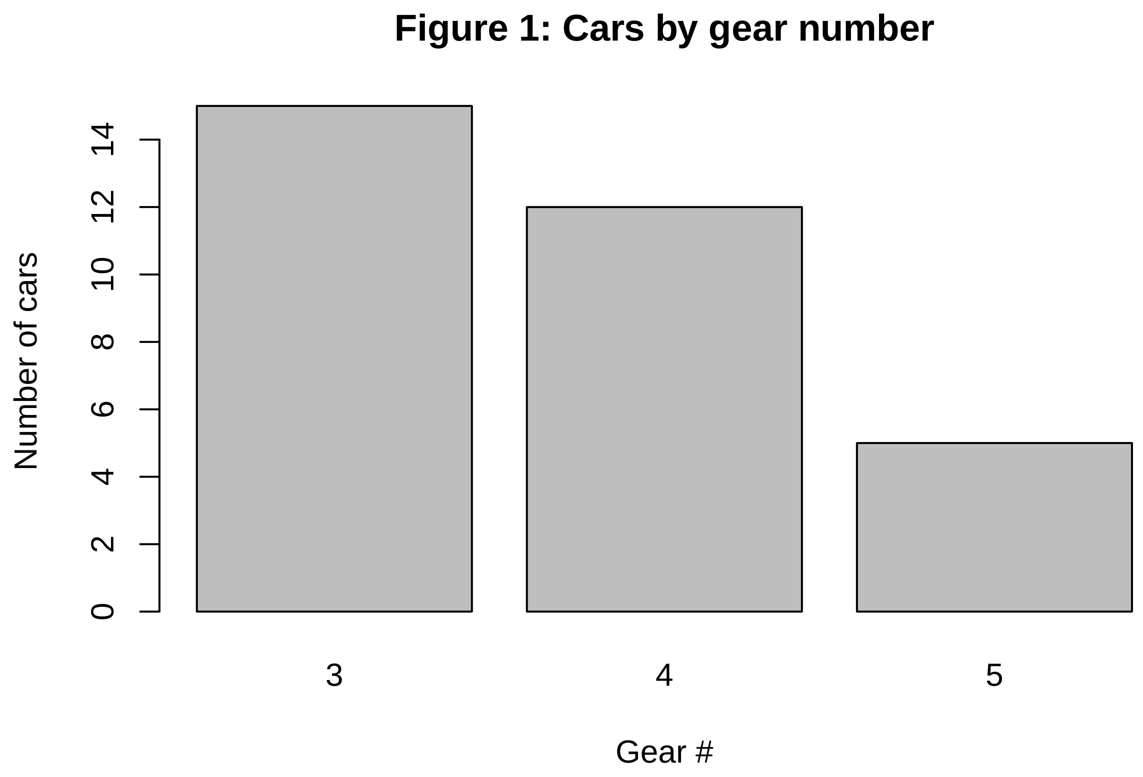
```
barplot(gear_table, main="Figure 1: Cars by gear number",
    xlab="Gear #", ylab= "Number of cars")
```



AYCS, commands in R can go over two lines. What's important is that thy are all in the same parenthetical brackets (). Using multiple lines is helpful when you have a particularly long command but you want to be able to see it all at once without scrolling over, although in theory every command can be executed on a single line if you are willing to scroll over to see the details of the command.

Finally, R allows you to use parentheses within parentheses, basically allowing you to execute a series of sequential commands on a single line. This can make coding much, more more simple and efficient, because instead of 1) writing multiple lines of code, 2) preparing the data object, and, 3) performing a command on that data object, you can do it all on one line. In the case above, instead of spending time and coding space creating a dataset of summary statistics, we can create it and then run the bar plot all at once, essentially combining the two commands above.

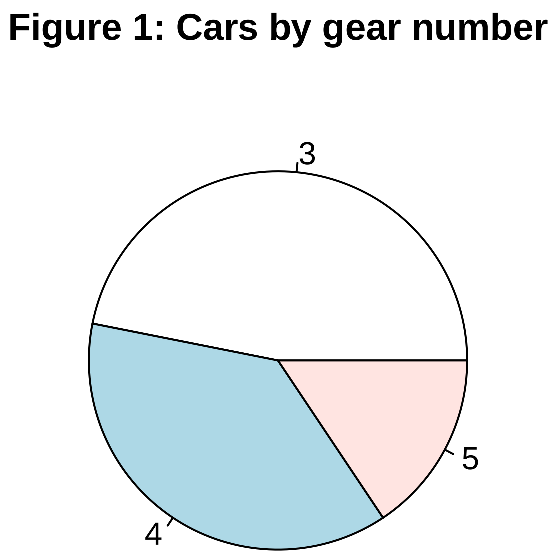
```
barplot(table(mtcars_data$gear),
        main="Figure 1: Cars by gear number", xlab="Gear #", ylab= "Number of cars")
```



AYCS, instead of creating **gear_table** and then running a separate command to create a bar plot on **gear_table**, we create **gear_table** within the *barplot* command. This works because *table(mtcar_data$gear)* is **gear_table**.

With the basic coding in place, it is easy to switch how we show our data by switching the graph type. For example, if we change out the *barplot* command with the *pie* command and take out the xlabel and ylabel options (since pie tables don't have x and y labels), we can visualize the same thing with a pie chart.

```
pie(table(mtcars_data$gear),
        main="Figure 1: Cars by gear number")
```



Once again, if you Google "Pie charts in R" or a similar phrase, you will see many different ways that you can customize this graph to make it look exactly how you want it to.