Introduction to Coding, IDE, and Quarto

Why are we coding?

To actually make use of what you learn in your statistics classes, you need to:

- Read in, store, manipulate, transform, subset, summarize, and visualize data
  - Not just numeric data, but also characters and strings
- Use and write your own algorithms to fit models to data
  - Assess how long applying methods to data might take
  - Use and write functions that facilitate generalization
- Simulate random variables

Keeping in mind, **any job you do that involves data analysis will require you to explain what you have done to people who know less about the data and statistical methods than you do**.

Why are we coding in R?

- Compared to Matlab, SAS, and Stata, it's free!
- Compared to Python it has:
  - A maintained, central repository of statistical software
  - A well developed set of data cleaning, manipulation, and visualization functions that are popular for their readability

Why RStudio?

Lets you simultaneously view:

- Your code files (where you keep a record of your code that can reproduce what you have done)
- The console (where the code runs)
- Plots
- Help files

Bonus: Gives you some point and click buttons that can help when you're stuck.

Why Quarto?

It makes it easier for you to produce **reproducible** summaries of data that can be understood more easily by others and future you (who sometimes understands less about what you did than others).

Why are we compiling to `.qmd` to `.pdf`?

- Standardized presentation
- Well integrated with LaTeX, which facilitates integrating bits of math into the text

What will we do today?

- Make a plot
- Make a table
- Reference a value from `R` in our presentation

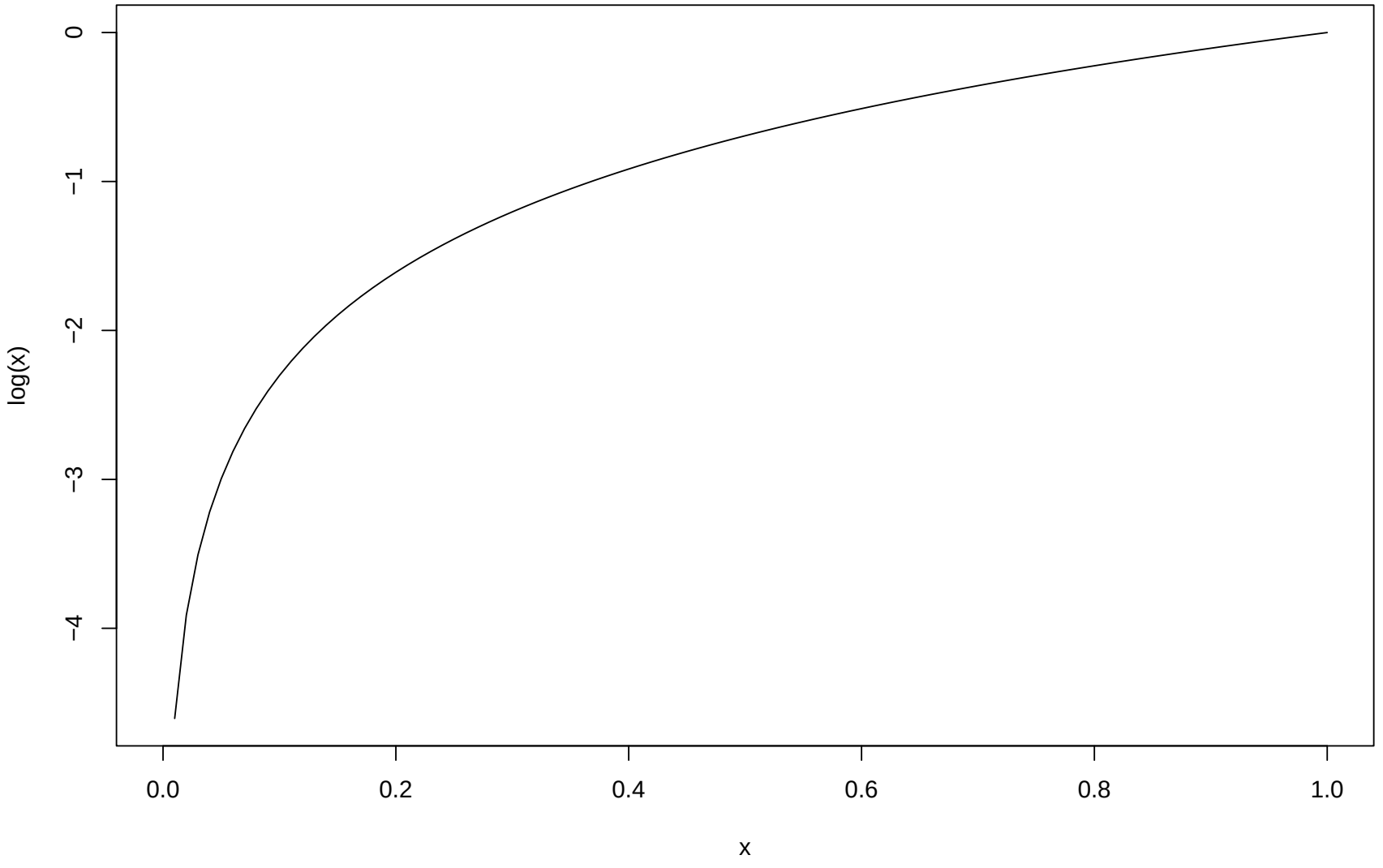
Plotting a Simple Function

```
curve(log)
```

`curve` is a function that takes another function, here `log`, as its first **argument**, and creates a plot of the function that as been provided.

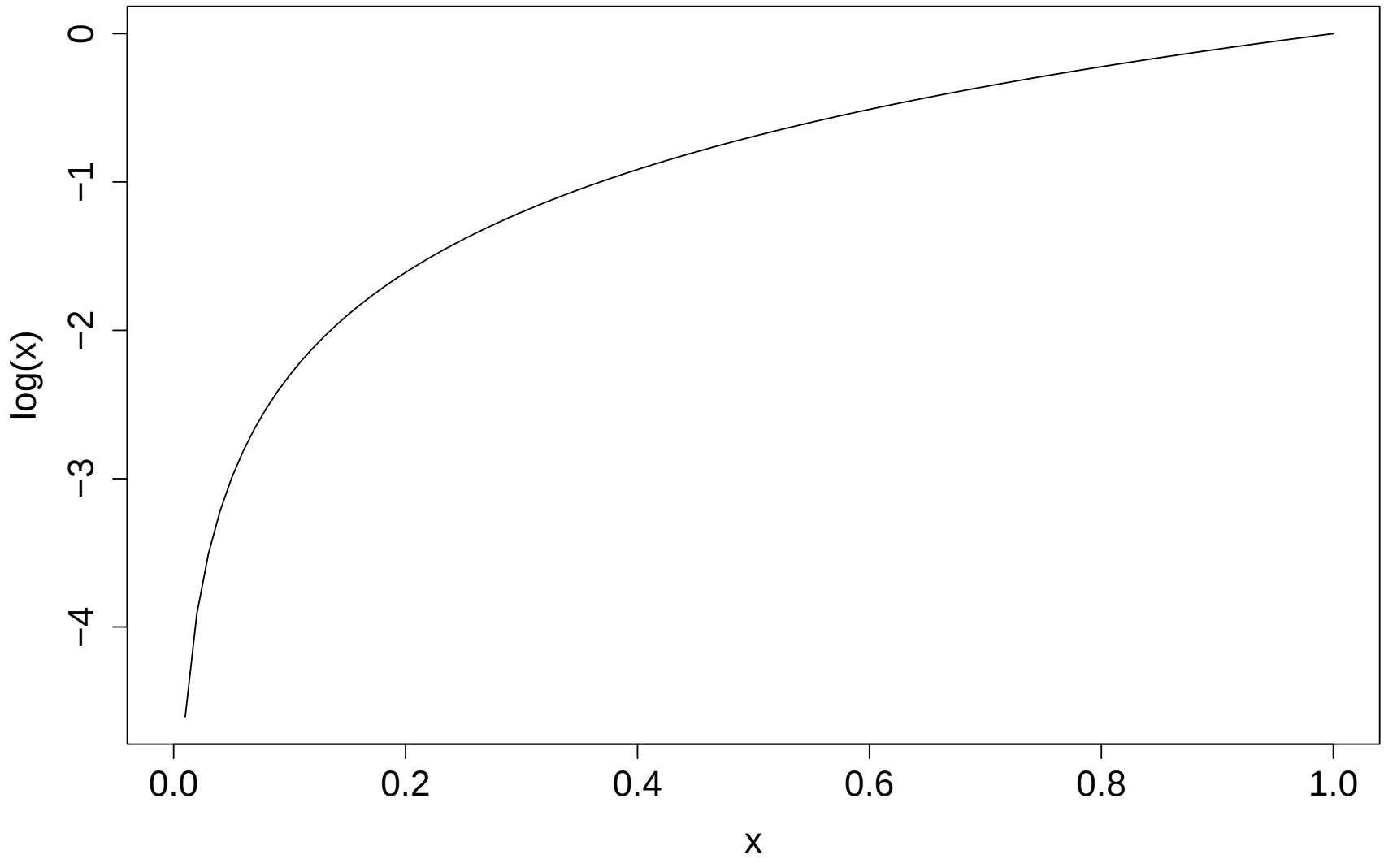
Plotting a Simple Function

```
curve(log)
```



Improving Legibility

```
curve(log, cex.lab = 1.5, cex.axis = 1.5)
```



Improving Legibility

```
curve(log, cex.lab = 1.5, cex.axis = 1.5)
```

`cex.lab` and `cex.axis` are **arguments** to the `curve` function. They are a specific type of **argument** that isn't always described in a help file because they can be used for a broad class of functions. Specifically, they are **graphical parameters**.

Default Arguments

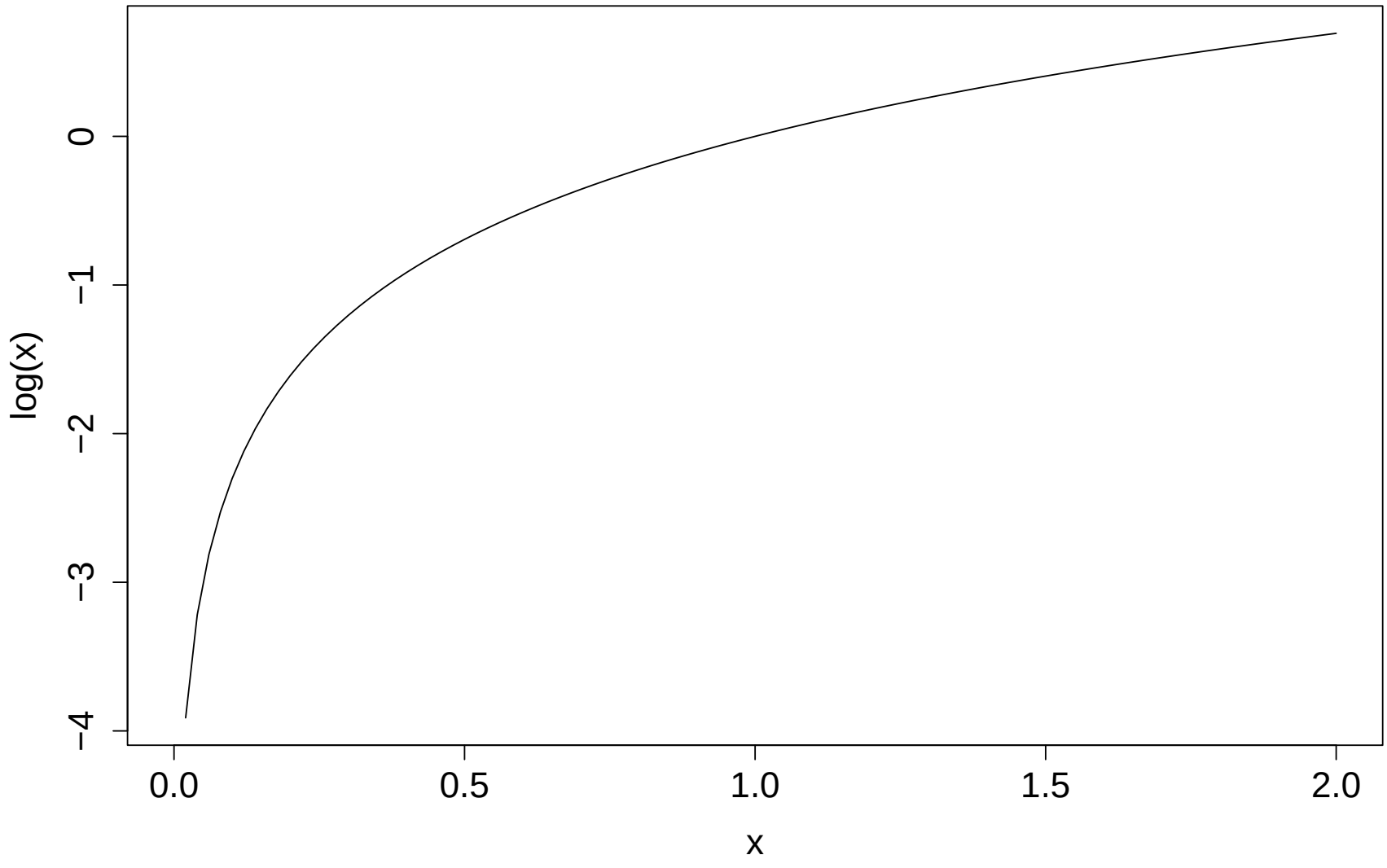
When we first called the `curve` command by typing `curve(log)`, we didn't specify `cex.lab` and `cex.axis`. But we still got a plot! How?

Arguments for functions have **defaults**, which are what is assumed if no value is provided.

The `curve` function has many other arguments that we have not specified, e.g. `from` and `to`.

Changing the Domain

```
curve(log, from = 0, to = 2,
      cex.lab = 1.5, cex.axis = 1.5)
```



How does curve work?

The curve function works by:

- Defining a sequence of values for the x-axis that are uniformly spaced between a minimum value and a maximum value

```
x <- seq(0, 1, length.out = 100)
```

- Evaluating a provided function at the sequence of x-values that have been constructed to obtain a set of values for the y-axis

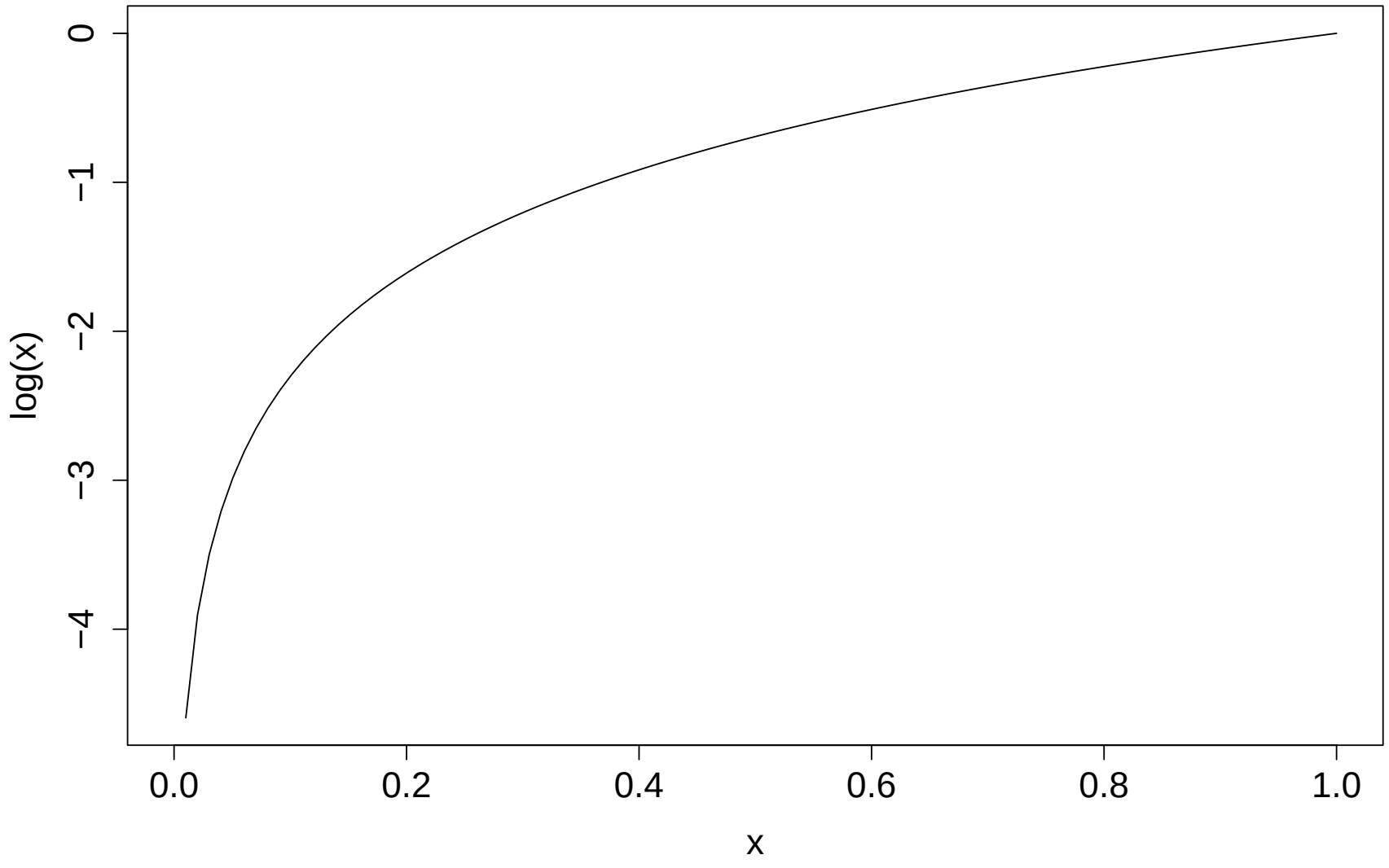
```
y <- log(x)
```

- Creating a curve by connecting the sequence of points corresponding to the values for the x- and y-axis as described in the previous steps

```
plot(x, y, type = "l", xlab = "x", ylab = "log(x)",
     cex.lab = 1.5, cex.axis = 1.5)
```

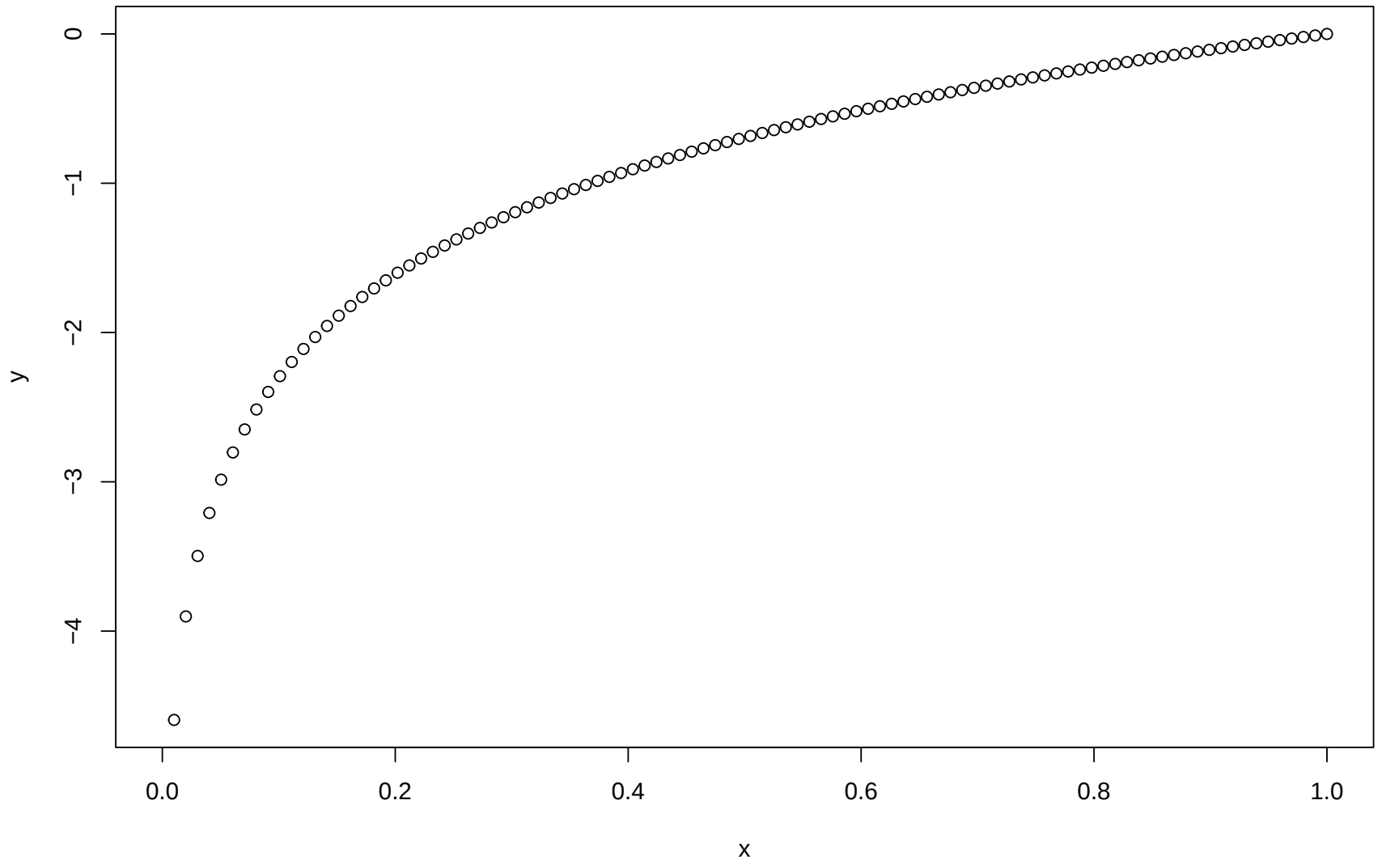
How does curve work?

```
plot(x, y, type = "l", xlab = "x", ylab = "log(x)",
     cex.lab = 1.5, cex.axis = 1.5)
```



Backing Up

```
plot(x, y)
```



Creating a Variable

The first thing we did was create a variable, `x`!

```
x <- seq(0, 1, length.out = 100)
```

We call `<-` the **assignment operator**. It assigns the variable `x` to refer to the output of a function, `seq(0, 1, length.out = 100)`.

It can be tempting to replace `<-` with `=`. Sometimes this will work and it can seem more parsimonious. Because it does not always work, sticking with `<-` is recommended. Type `help(assignOps)` into the console to learn more if you are curious.

Constructing an Equally Spaced Sequence

```
x <- seq(0, 1, length.out = 100)
```

The `seq` function with first argument `0`, second argument `1`, and `length.out = 100` creates a **vector** with 100 elements, where the value of the k-th element is $k/100$.

It creates an evenly spaced sequence of `length.out` numbers between the first argument and the second argument.

Note that if an argument isn't named, then the order it appears in determines what it corresponds to.

What is a vector???

```
str(x)
```

```
 num [1:100] 0 0.0101 0.0202 0.0303 0.0404 ...
```

Applying a Function to a Vector

R has many built in basic math functions that, when applied to a vector, apply the function elementwise and return a vector.

```
y <- log(x)
```

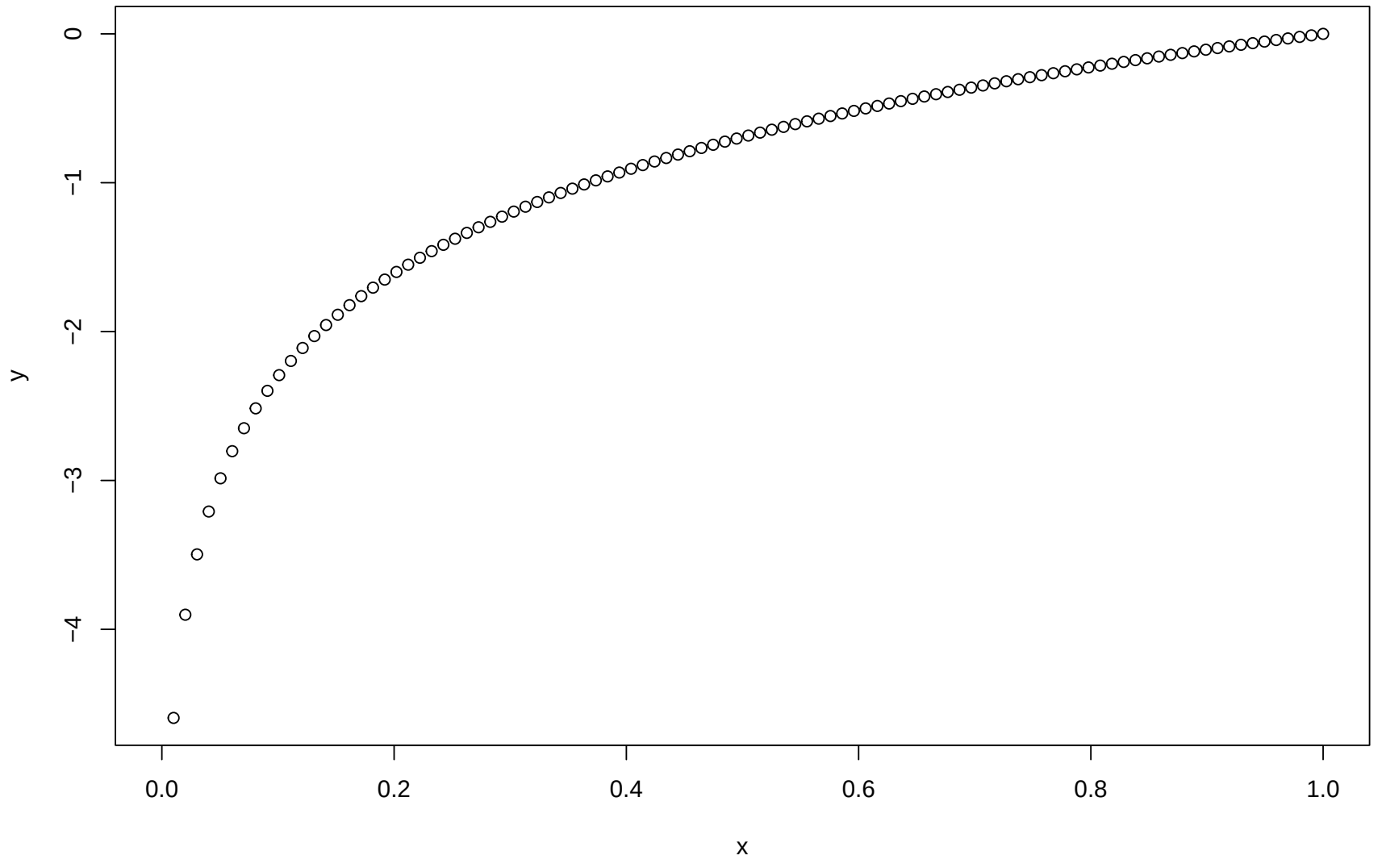
Here, the log function returns a new vector obtained by taking each element of x and computing its logarithm.

```
str(y)
```

```
 num [1:100] -Inf -4.6 -3.9 -3.5 -3.21 ...
```

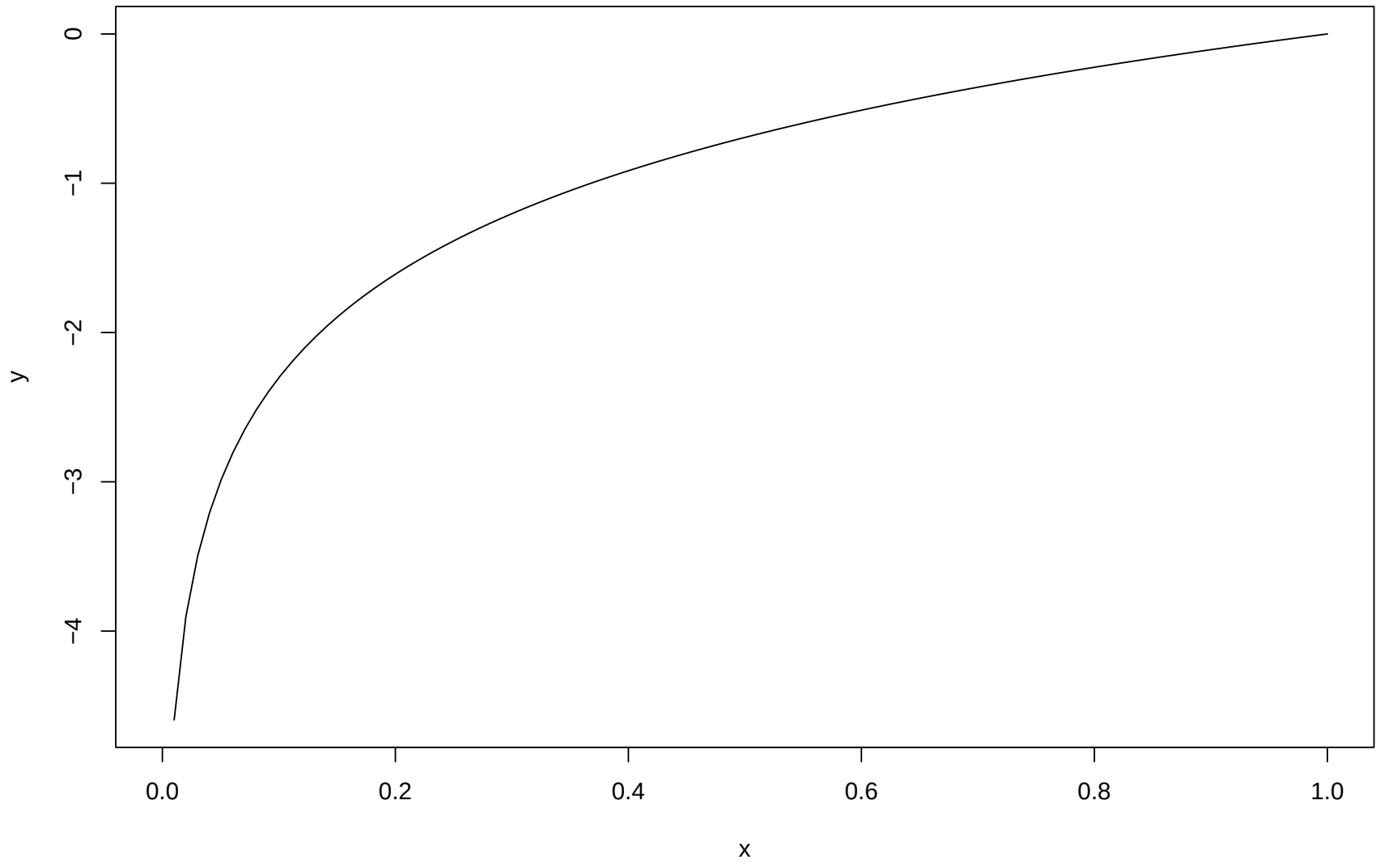
Using `plot` to Create a Plot

```
plot(x, y)
```



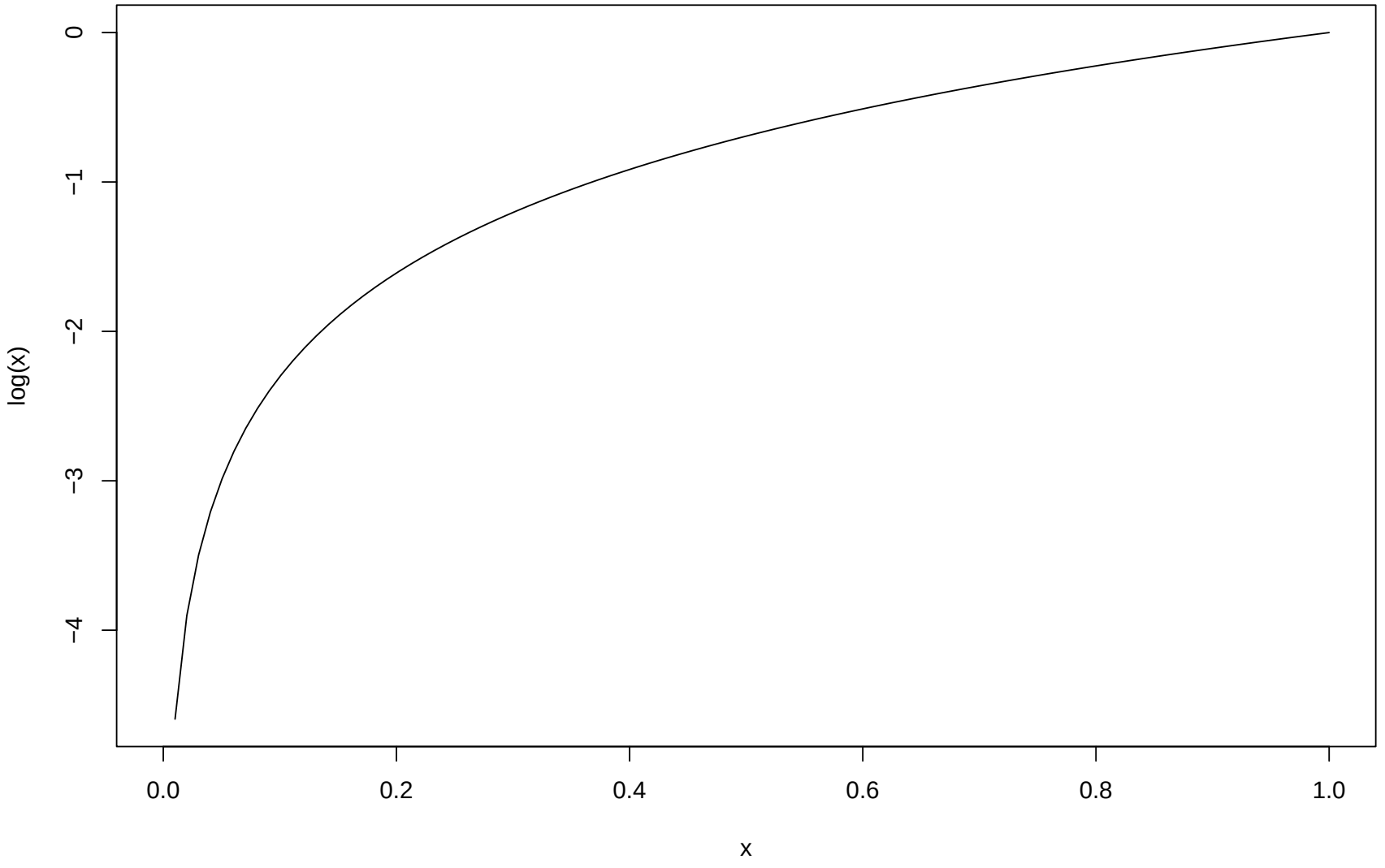
Connecting Points into a Line

```
plot(x, y, type = "l")
```



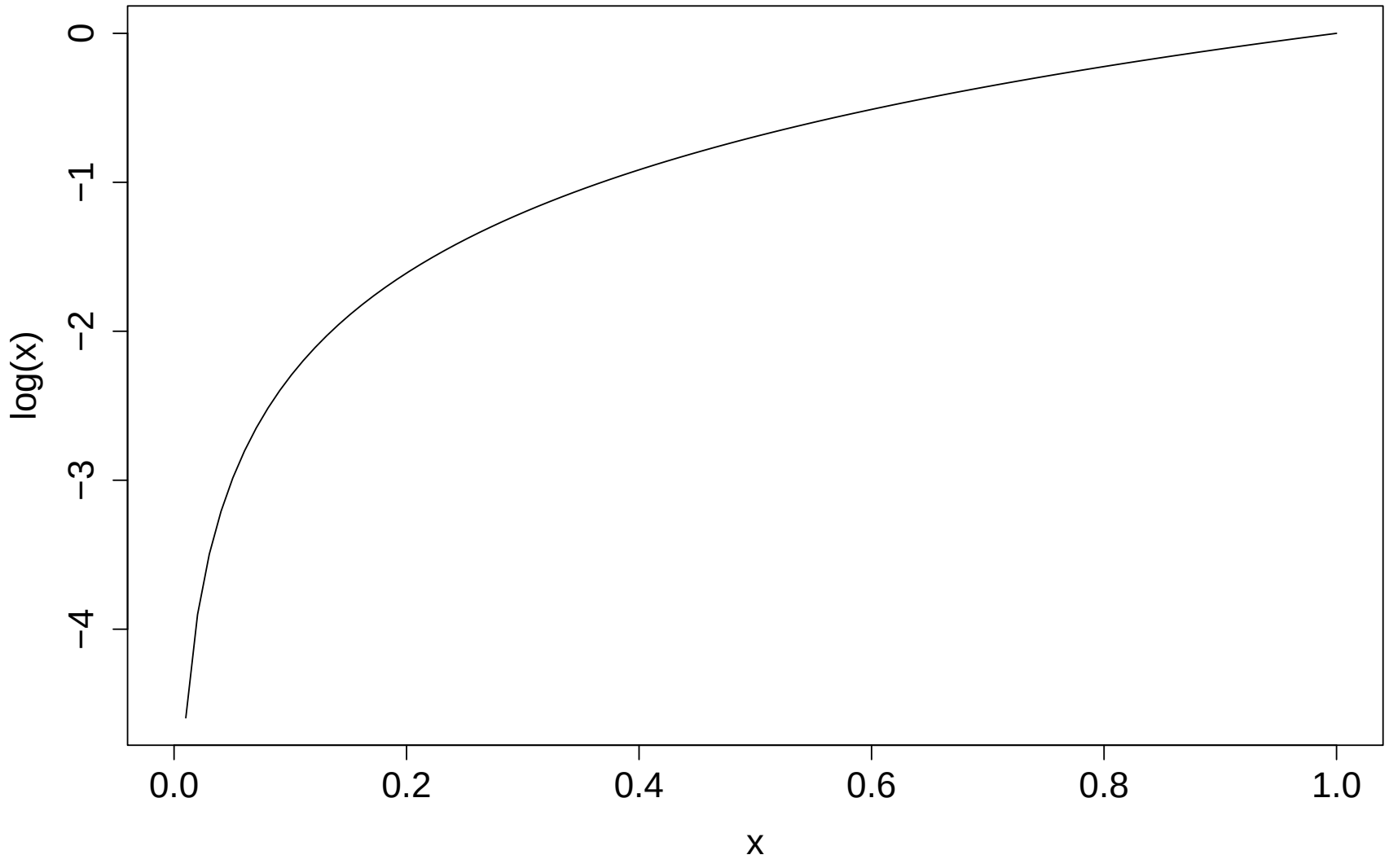
Changing Axis Labels

```
plot(x, y, type = "l", xlab = "x", ylab = "log(x)")
```



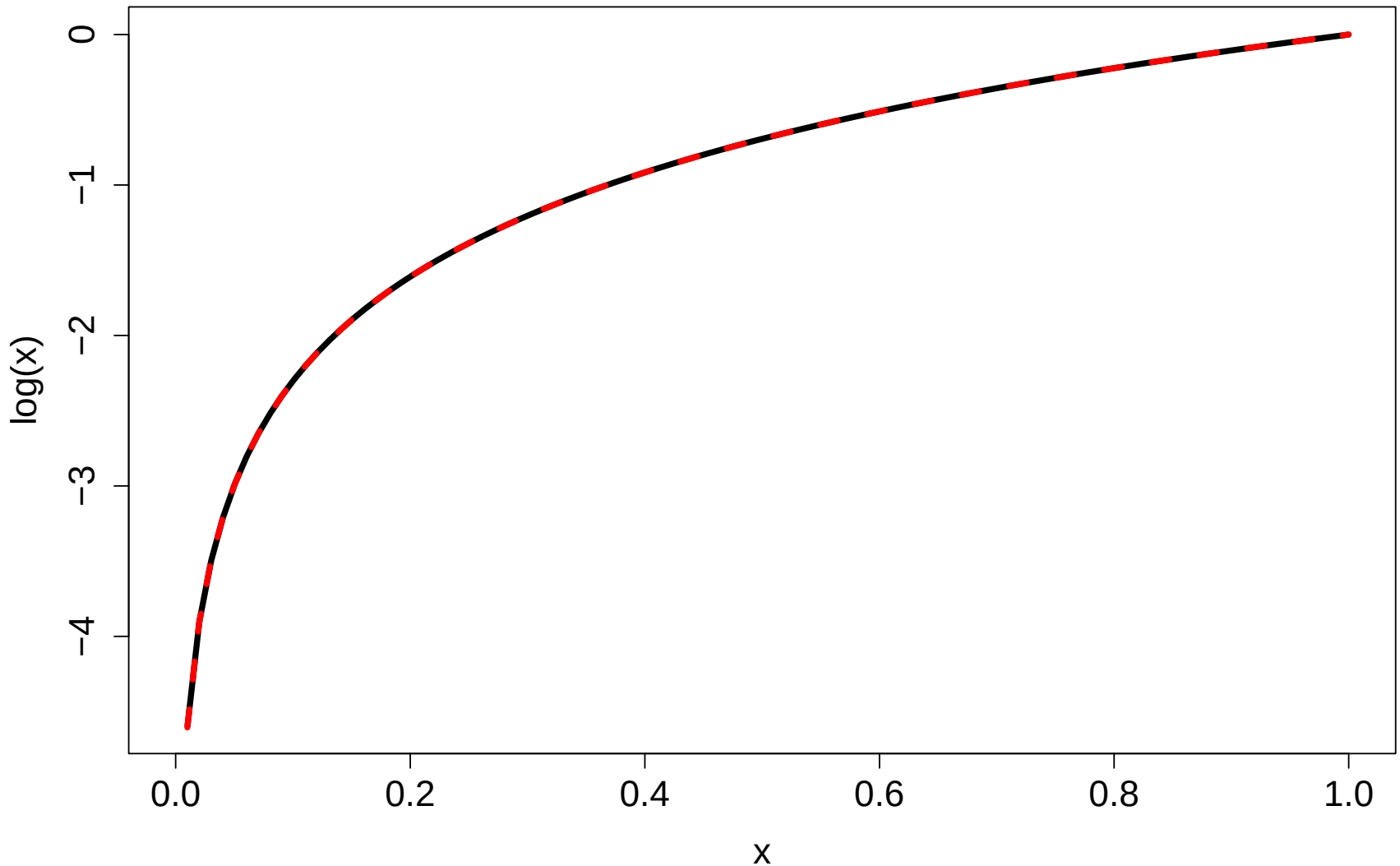
Making It Legible

```
plot(x, y, type = "l", xlab = "x", ylab = "log(x)",
     cex.lab = 1.5, cex.axis = 1.5)
```



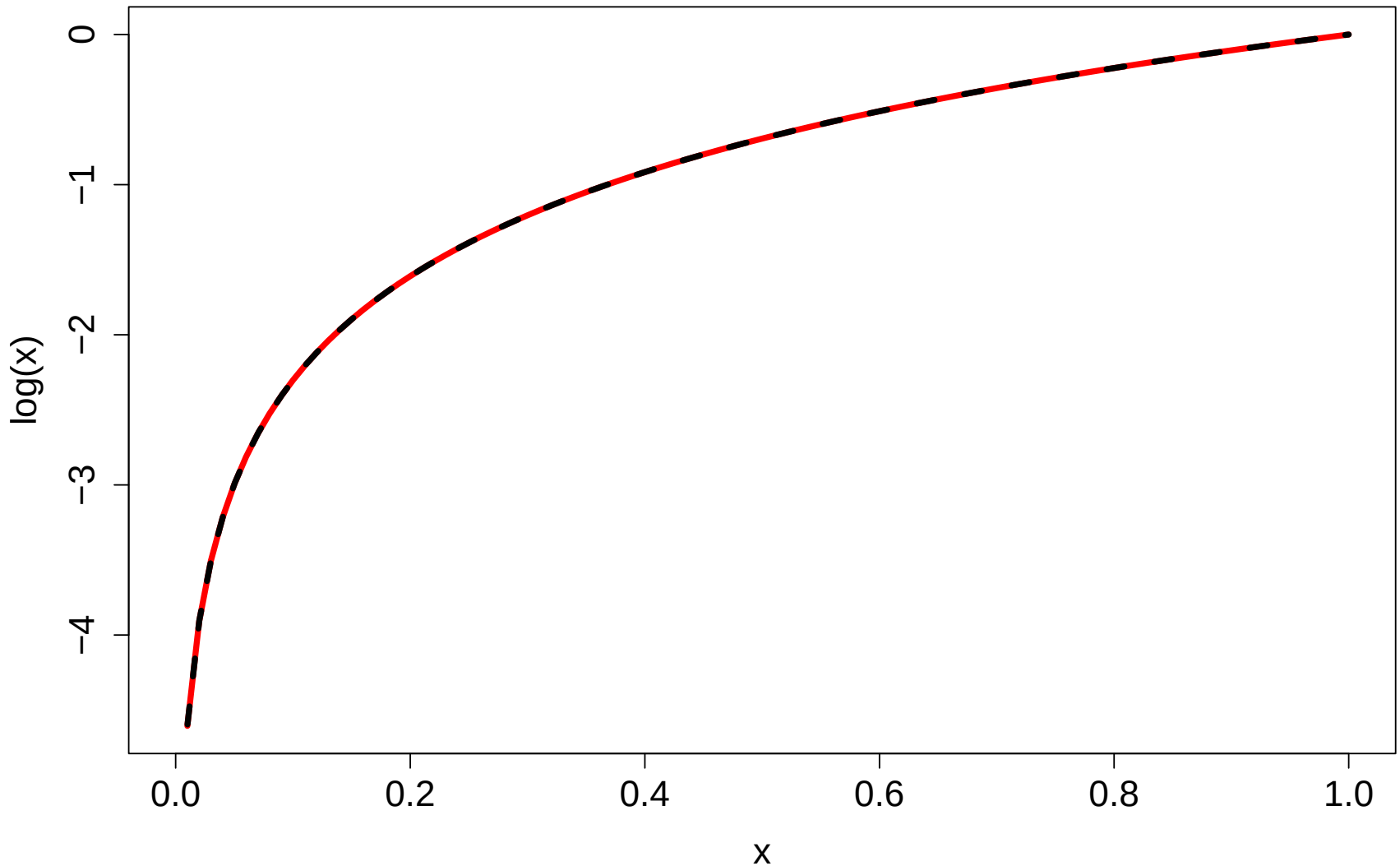
Comparing to curve

```
plot(x, y, type = "l", xlab = "x", ylab = "log(x)",
     cex.lab = 1.5, cex.axis = 1.5, lwd = 4)
curve(log(x), add = TRUE, col = "red", lty = 2, lwd = 4)
```



Comparing to curve

```
curve(log(x), col = "red",
      cex.lab = 1.5, cex.axis = 1.5, lwd = 4)
lines(x, y, type = "l", lty = 2, lwd = 4)
```



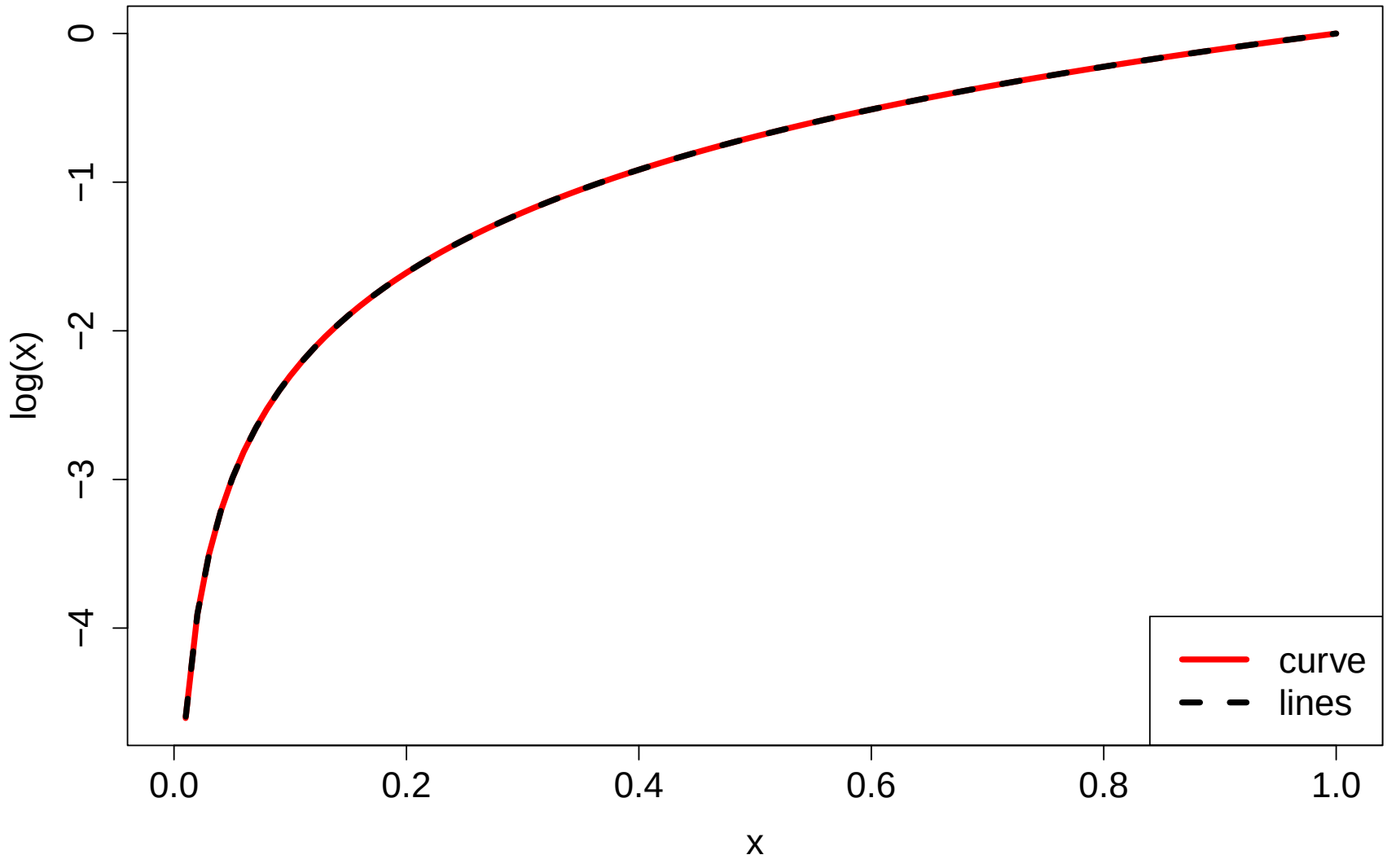
Adding a Legend

```
curve(log(x), col = "red",
      cex.lab = 1.5, cex.axis = 1.5, lwd = 4)
lines(x, y, type = "l", lty = 2, lwd = 4)
legend("bottomright",
       lty = c(1, 2),
       lwd = 4, col = c("red", "black"),
       legend = c("curve", "lines"), cex = 1.5)
```

Notes:

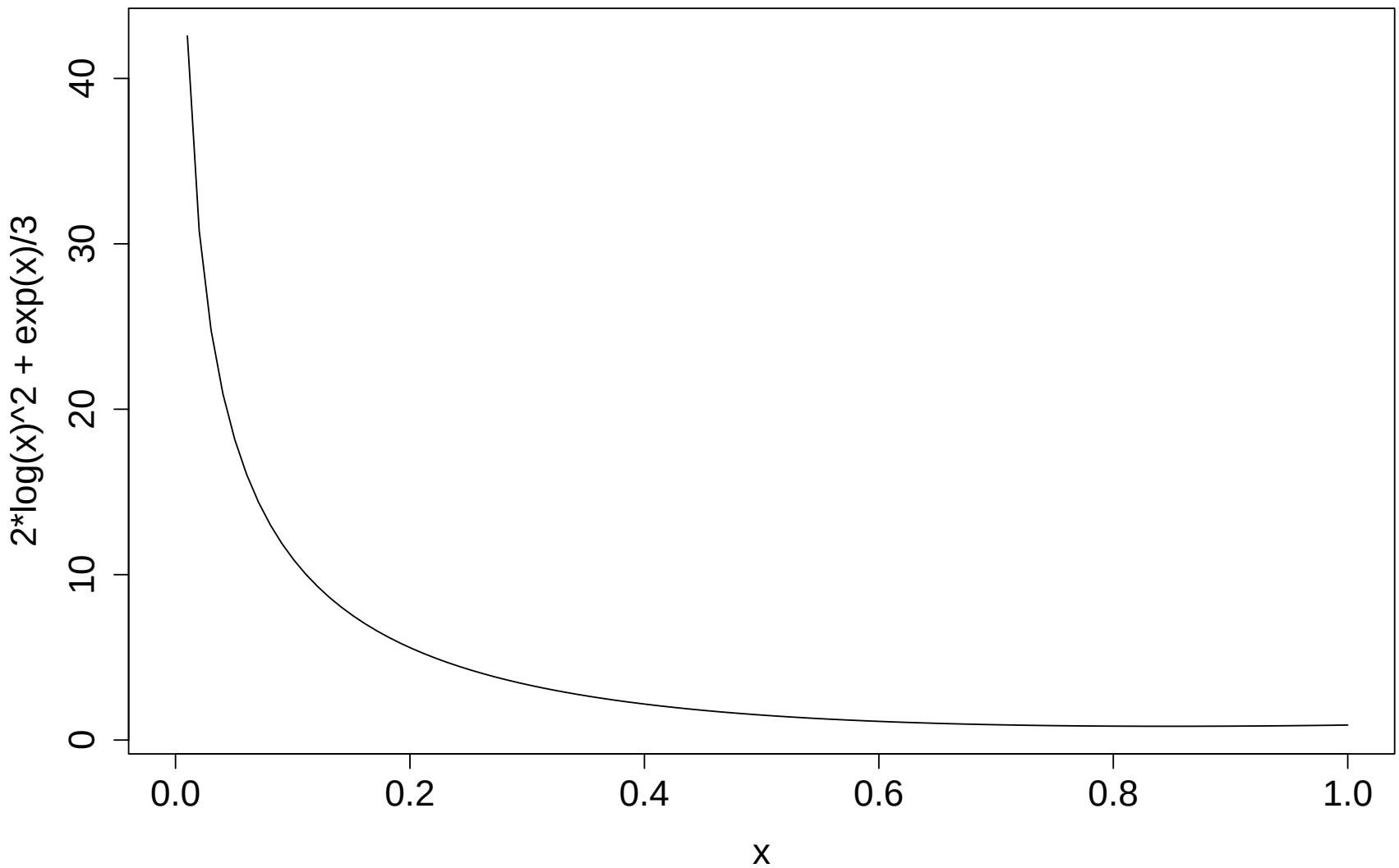
- The default line type is a solid line, which is `lty = 1`
- c stands for **concatenate**. It is a function that joins consecutive elements together as a vector

Adding a Legend



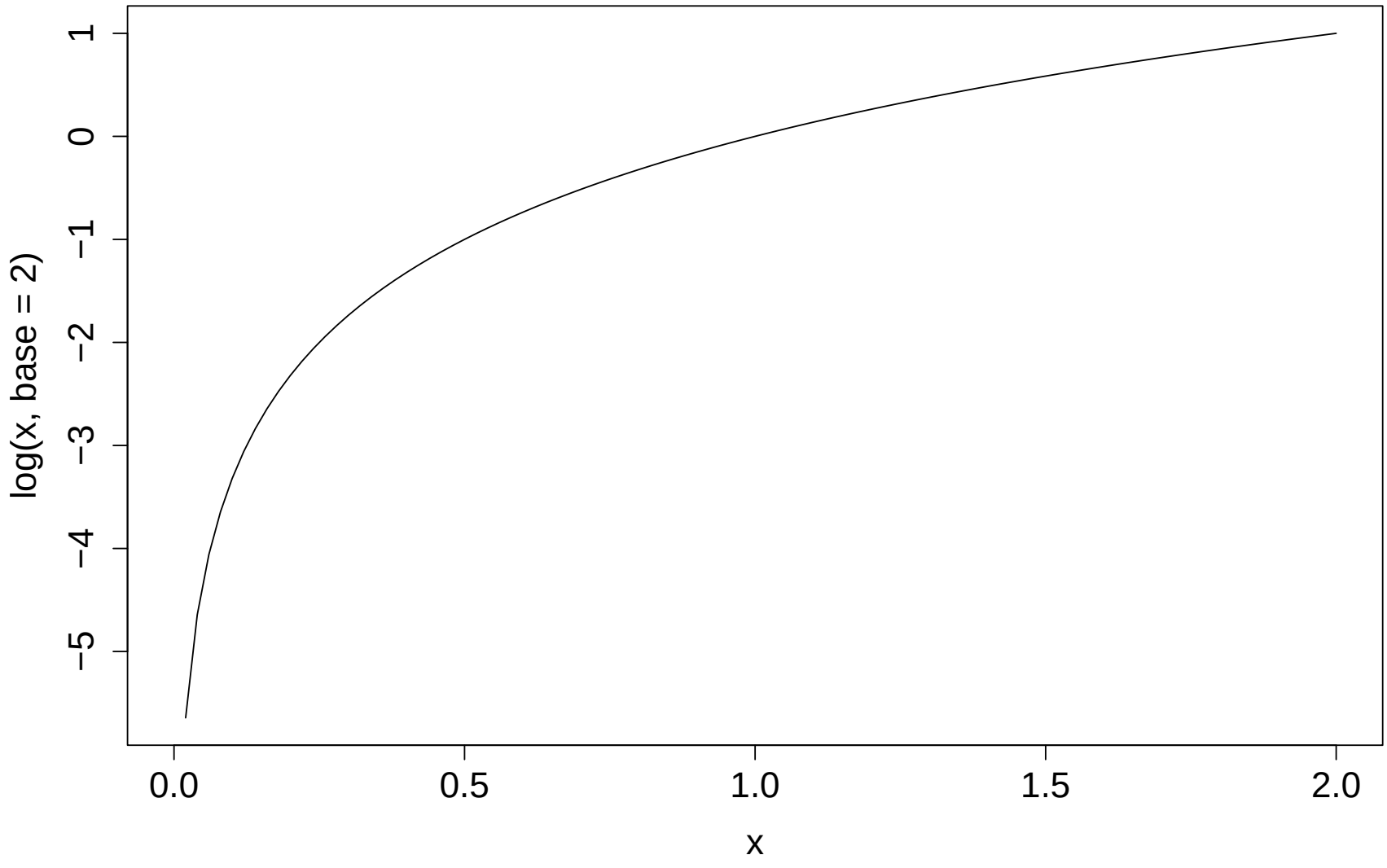
Making A More Complicated Plot

```
plot(x, 2*y^2 + exp(x)/3, type = "l", xlab = "x",
     ylab = "2*log(x)^2 + exp(x)/3",
     cex.lab = 1.5, cex.axis = 1.5)
```



Making A More Complicated Plot

```
curve(log(x, base = 2), from = 0, to = 2,
      cex.lab = 1.5, cex.axis = 1.5)
```



Making a Table

Making a table starts with storing a table. In R, tables are stored as matrices, arrays, or data frames. We'll talk about them in detail later. For now, we'll use a matrix.

```
Y <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)
```

```
str(Y)
```

```
 num [1:2, 1:2] 1 2 3 4
```

```
Y
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Making a Table

```
colnames(Y) <- c("Column 1", "Column 2")
row.names(Y) <- c("Row 1", "Row 2")
```

```
knitr::kable(Y)
```

	Column 1	Column 2
Row 1	1	3
Row 2	2	4

Referencing a Value from R

We can reference individual elements of an matrix by using brackets and specifying the desired row and column, e.g. `Y[1, 1]` will return the element in the first row and first column of `Y`.

Based on what we defined earlier, $Y_{11} = 1$, $Y_{12} = 3$, $Y_{21} = 2$, and $Y_{22} = 4$.

Note - these values were directly pulled from `R`, look at the `.qmd` file associated with this `.pdf` to see how!

Takeaways

Every plot you make should be:

- Legible
- Self-contained

Every table you make should be:

- Formatted, not just printed `R` output

Avoid copying and pasting results whereever possible!