Week 4 Module 5: Histograms

CSCI E-5a: Programming in R

Let's clear the global computing environment:

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

```
load( "Module 5 R Objects.Rdata" )

ls()
```

```
## [1] "example.1.data"    "exercise.3.1.data"
```

Module Overview and Learning Outcomes

Hello! And welcome to Module 5: Histograms.

In this module we'll explore a new visualization method known as the *histogram*.

- In section 1, we'll learn to construct a basic histogram.
- In section 2, we'll see how to modify the basic histogram.

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

- Explain how a histogram displays the distribution of values in a numeric vector.
- Create a basic histogram for a numeric vector.
- Modify the basic histogram: adding titles, adjusting the color of the bars, and specifying the number of breaks.

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

- `hist()`

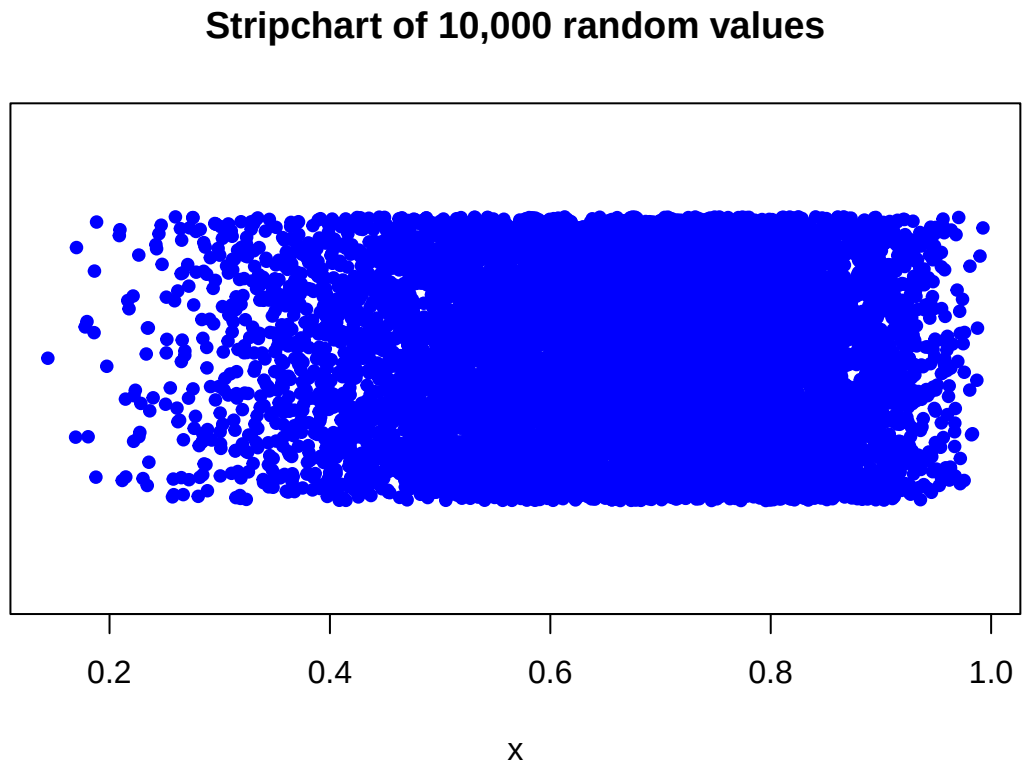
Section 1: Histogram Basics

> **Main Idea:** *Let's construct a histogram*

In this section, we'll learn to construct a basic histogram.

Let's take a look at the stripchart of `example.1.data`:

```
stripchart(
    example.1.data,
    main = "Stripchart of 10,000 random values",
    xlab = "x",
    col = "blue",
    ylim = c(0,2),
    method = "jitter",
    jitter = 0.6,
    pch = 19,
    cex = 0.8
)
```



This isn't really useful, because there are so many data points that the stripchart loses resolution.

Instead, a *histogram* will enable us to visualize the distribution of this data, and it will scale well as the sample size becomes large.

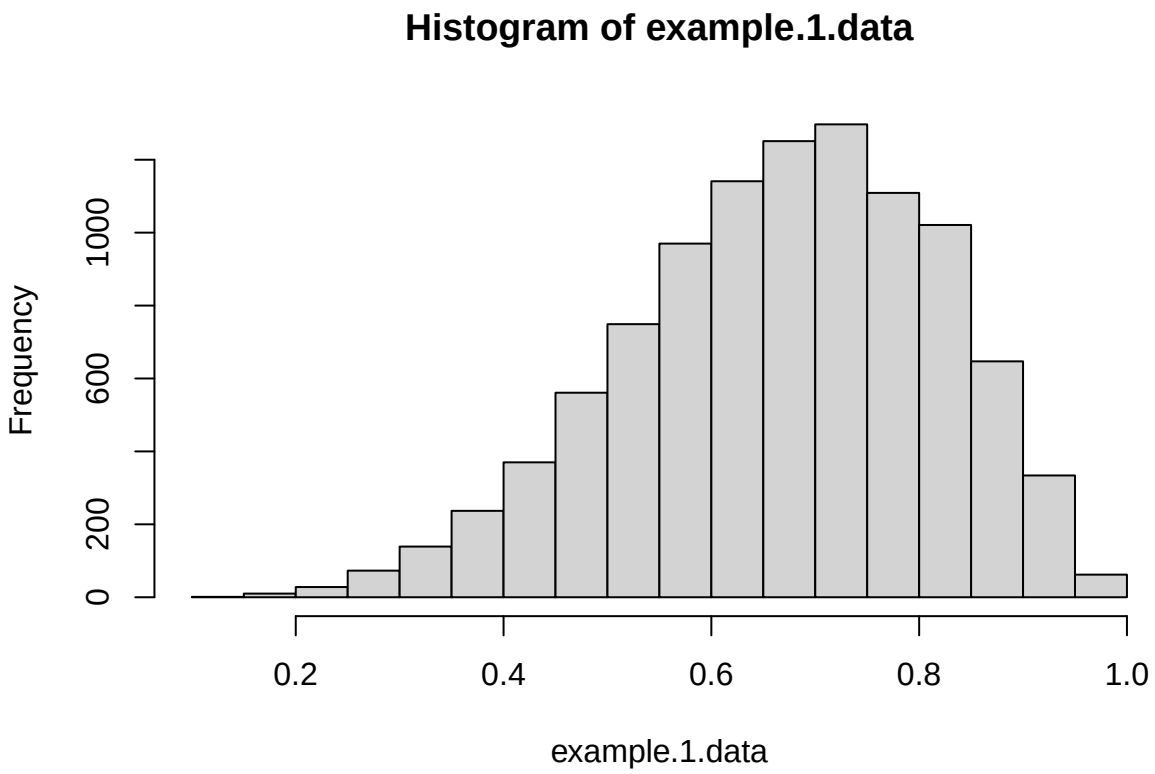
With a histogram, the range of numeric data is split up into a number of subintervals.

R counts how many elements of the data are contained in each subinterval.

R then constructs a chart with bars, where the width of the bar is the subinterval and the height of the bar is the relative proportion of elements contained in the interval.

This is the most basic form of a histogram:

```
hist( example.1.data )
```



It's useful to see the most basic histogram, just to get an idea of the fundamental structure of the graph.

So that's how to construct a histogram.

Now let's see how to modify this display.

Exercise 3.1: Basic Histogram

Create a basic histogram for the values in `exercise.3.1.data`.

Solution

Section 2: Modifying the Histogram

> **Main Idea:** *We can modify the basic histogram*

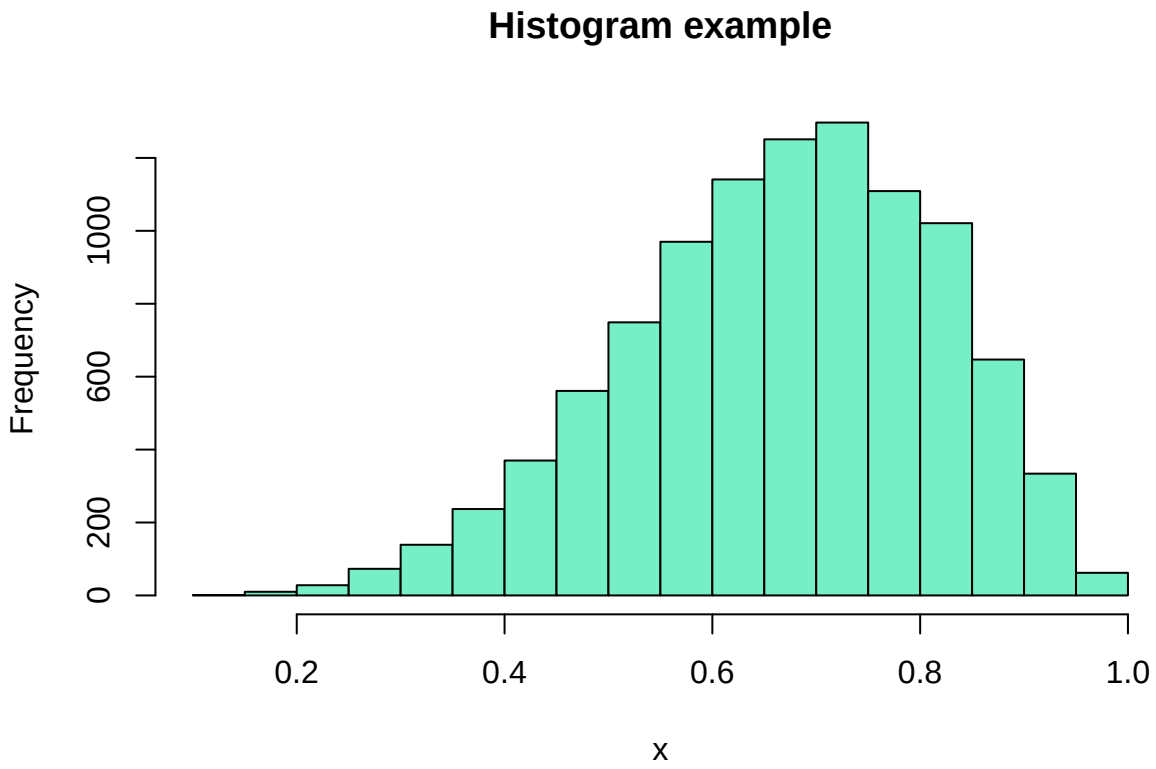
In this section, we'll see how to modify the basic histogram.

Of course, we can immediately make our basic histogram nicer by supplying a main title, as well as titles for the x and y-axes.

We can also color the bars using the `col` option:

```
hist(
    example.1.data,
    main = "Histogram example",
```

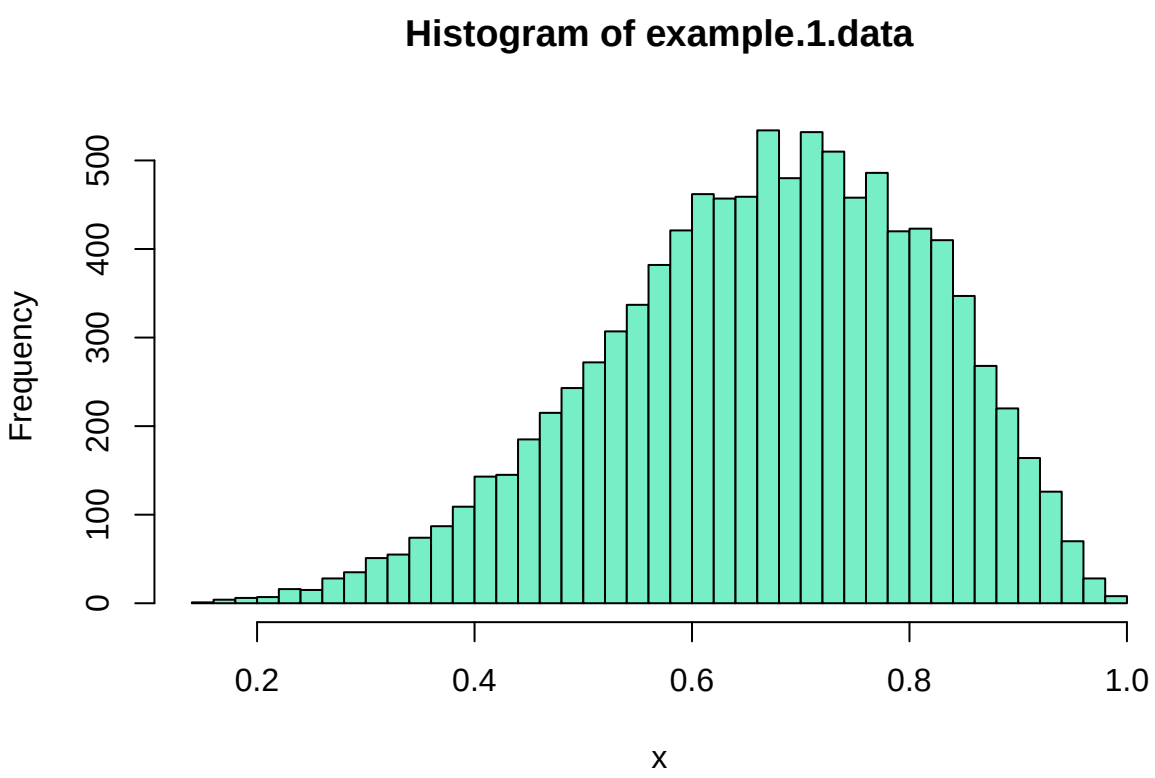
```
    xlab = "x",
    ylab = "Frequency",
    col = "aquamarine2"
)
```



Notice that the height of the bars reflects the *absolute* frequency, that is, the total number of observations in the vector that fall into the particular bin.

We can control the number of bins by using the `breaks` option:

```
hist(
    example.1.data,
    main = "Histogram of example.1.data",
    xlab = "x",
    ylab = "Frequency",
    col = "aquamarine2",
    breaks = 50
)
```



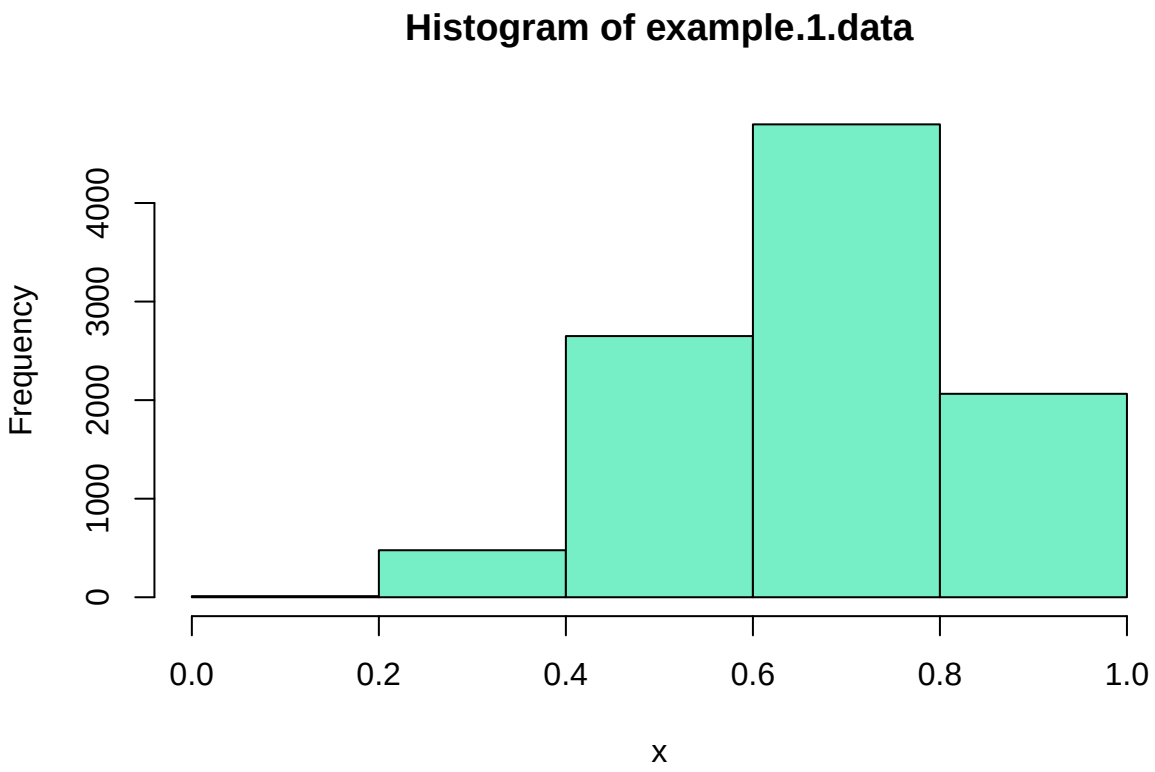
Now you can see that the histogram is much more fine-grained, with a higher resolution than before, because we are using more bins.

In general, when you set the number of bins by assigning a positive integer to the `breaks` option, R will decide the boundaries of the bins.

In fact, R will also take your specification of the number of breaks as a suggestion, and it will determine how many bins there will actually be.

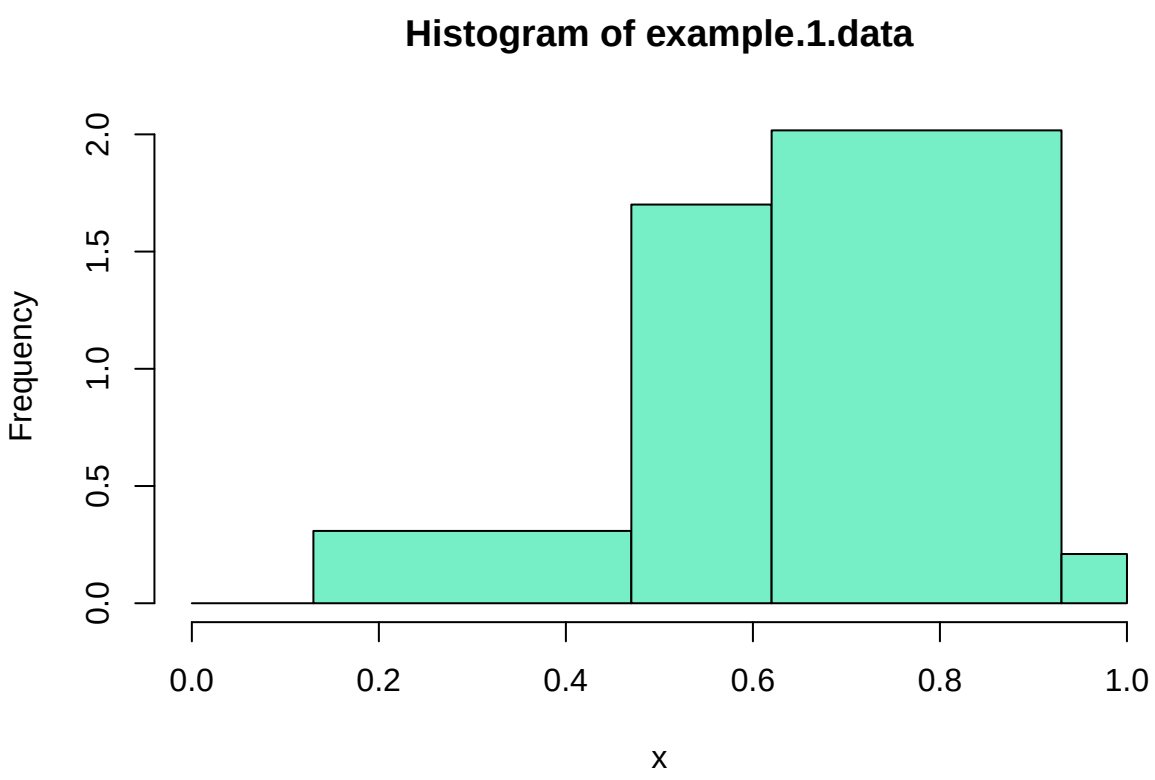
If you want complete control over the bins, you can pass a vector of endpoints to the `breaks` option:

```
hist(
    example.1.data,
    main = "Histogram of example.1.data",
    xlab = "x",
    ylab = "Frequency",
    col = "aquamarine2",
    breaks = c(0, 0.2, 0.4, 0.6, 0.8, 1.0)
)
```



The bin sizes don't have to all be the same, and the cutpoints don't have to be "nice" values:

```
hist(
    example.1.data,
    main = "Histogram of example.1.data",
    xlab = "x",
    ylab = "Frequency",
    col = "aquamarine2",
    breaks = c(0, 0.13, 0.47, 0.62, 0.93, 1.0)
)
```

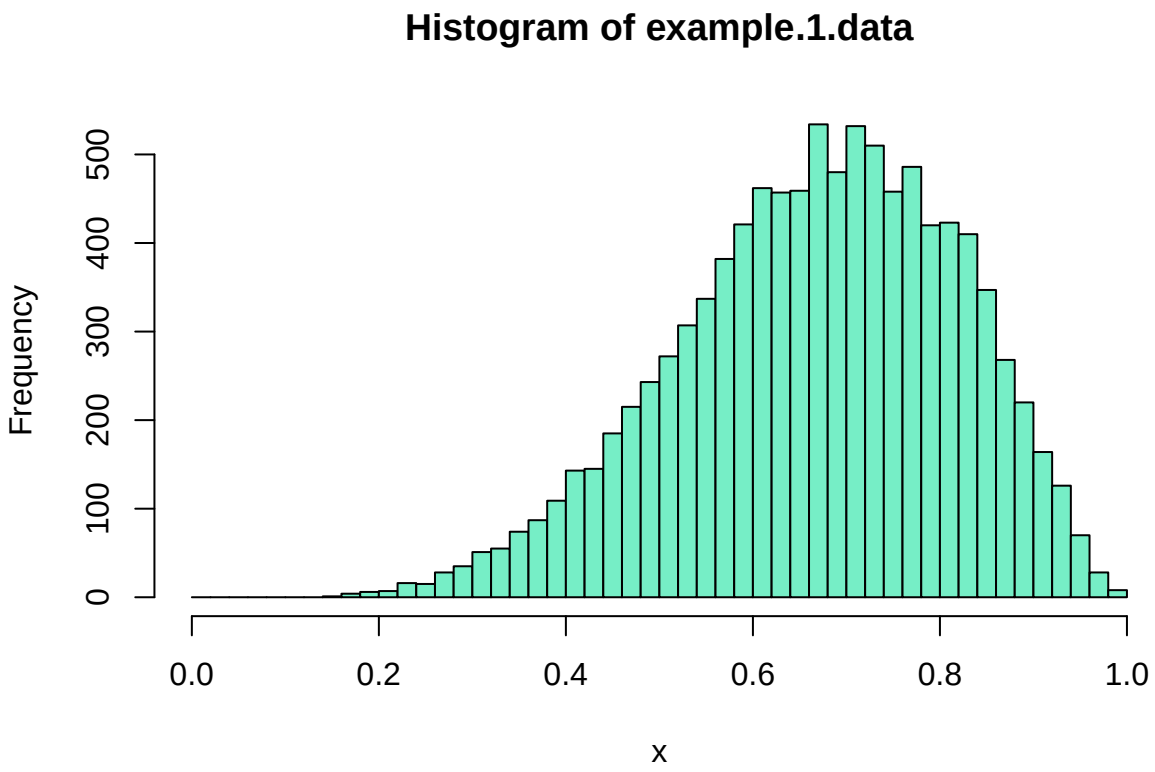


These are weird-looking graphs, and I don't think that they are particularly useful, but they do illustrate how flexible the `breaks` option can be.

The important point is that we can completely control the break points by passing a vector if we want to, and this process is completely general.

The `seq()` function is extremely useful for specifying break points:

```
hist(
    example.1.data,
    main = "Histogram of example.1.data",
    xlab = "x",
    ylab = "Frequency",
    col = "aquamarine2",
    breaks =
        seq( from = 0, to = 1.0, by = 0.02)
)
```



So that's how to modify the basic histogram.

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

Exercise 3.2: Basic Histogram

Construct a histogram for the numeric vector `exercise.3.1.data`. Include a main title and titles for the x-axis and y-axis, select a color for the bars, and explicitly specify the number of breaks.

Solution

Module Review

In this module we explored a new visualization method known as the *histogram*.

- In section 1, we learned to construct a basic histogram.
- In section 2, we saw how to modify the basic histogram.

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

- Explain how a histogram displays the distribution of values in a numeric vector.
- Create a basic histogram for a numeric vector.

- Modify the basic histogram: adding titles, adjusting the color of the bars, and specifying the number of breaks.

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

- `hist()`

All right! That's it for Module 5: Histograms.

In fact, that's all the content for Week 4: Iteration.

Now you can finish Problem Set 4.

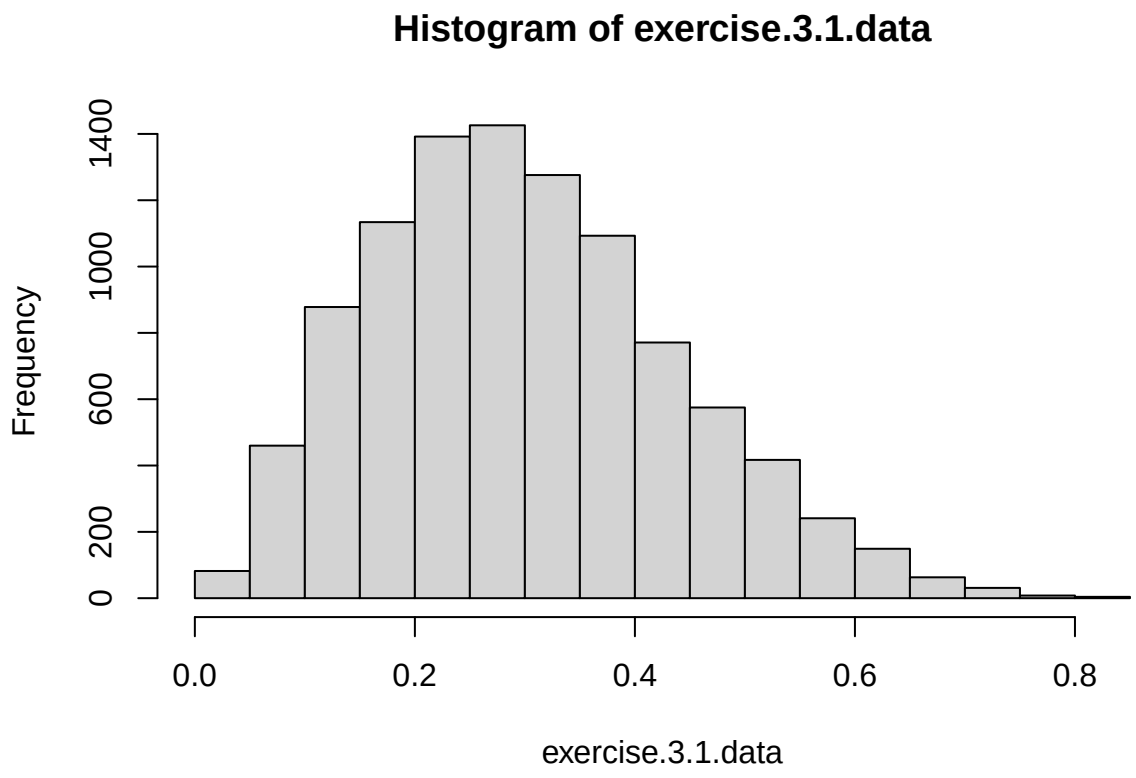
Solutions to the Exercises

Exercise 3.1: Basic Histogram

Create a basic histogram for the values in `exercise.3.1.data`.

Solution

```
hist( exercise.3.1.data )
```



Exercise 3.2: Fancy Histogram

Construct a histogram for the numeric vector `exercise.3.1.data`. Include a main title and titles for the x-axis and y-axis, select a color for the bars, and explicitly specify the number of breaks.

Solution

```
hist(
    exercise.3.1.data,
    main = "Histogram of exercise.3.1.data",
    xlab = "x",
    ylab = "Frequency",
    col = "plum2",
    breaks = 50
)
```