Generating and Plotting Data

Nicholas A. Del Grosso

July 22, 2018

Vector Generation

Hand-typing out vectors with the **c()** function will only get us so far–it's much nicer to ask R to generate vectors for us! Most of these random-generation functions start with the letter r (e.g. *rnorm()*, *runif()*, *rgamma()*, *rpoisson()*, *rweibull()*). Sequences can also be generated as a vector with *seq()*, and numbers and vectors can even be repeated with *rep()*.

Note: Some of these functions require more than one arguments. If you see an error message that says "argment '—' is missing, with no default", it means that you should put additional arguments into the function, seperated by commas.

5 Normally-distributed values (mean = 0, sd = 1):

```
rnorm(5)
```

```
## [1]  1.02681364  0.36762043  0.75450516 -0.05242568  0.56683540
```

5 Uniformly-distributed values between 0 and 1

```
runif(5)
```

```
## [1] 0.70568956 0.80753184 0.18165416 0.88285044 0.07213843
```

The number 2 repeated 6 times:

```
rep(2, 6)
```

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

The sequence from 3 to 8:

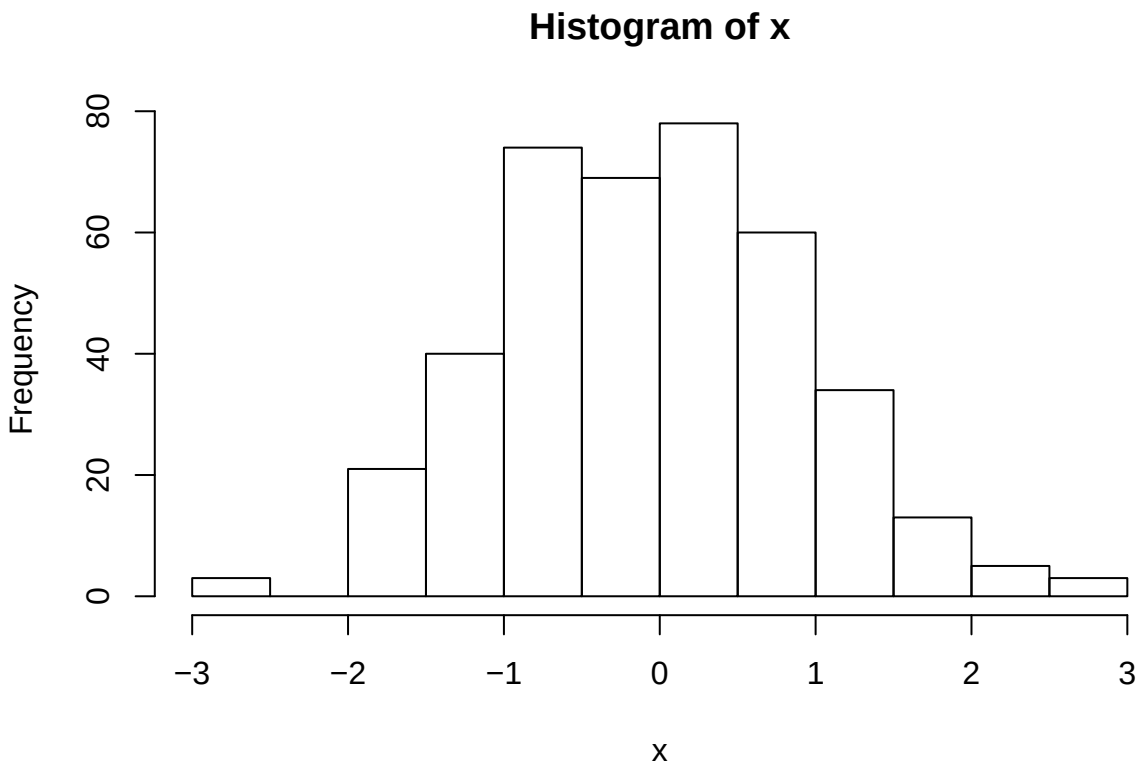
```
seq(3, 8)
```

```
## [1] 3 4 5 6 7 8
```

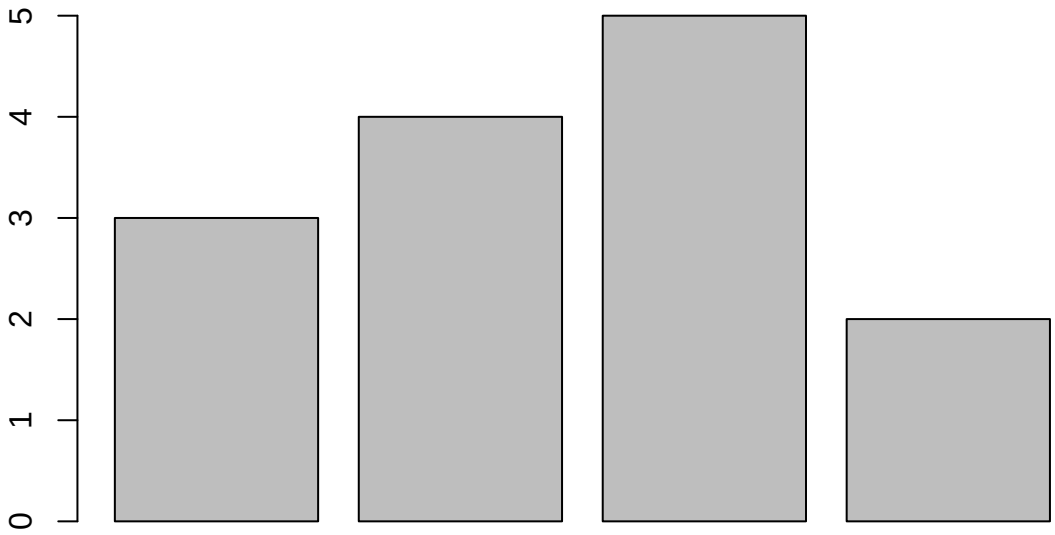
Plots

R also comes with plotting and grpahics packages, each type with their own plotting function. Some plot types have their own functions, like histograms (*hist()*) bar plots (*barplot()*), and scatter plots with smoothed curve fits (*scatter.smooth()*). These plots do several actions in a single function call: they make a new window, draw the axes, plot the data, and add axis labels:

```
x <- rnorm(400)
hist(x)
```

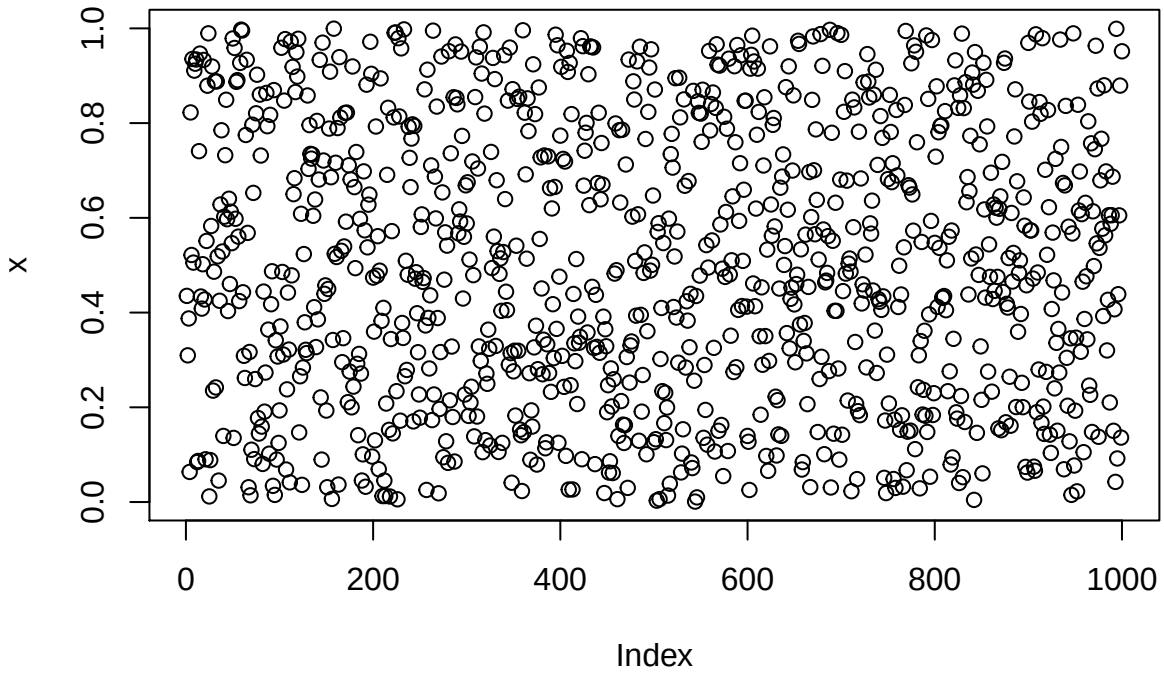



```
x <- c(3, 4, 5, 2)
barplot(x)
```

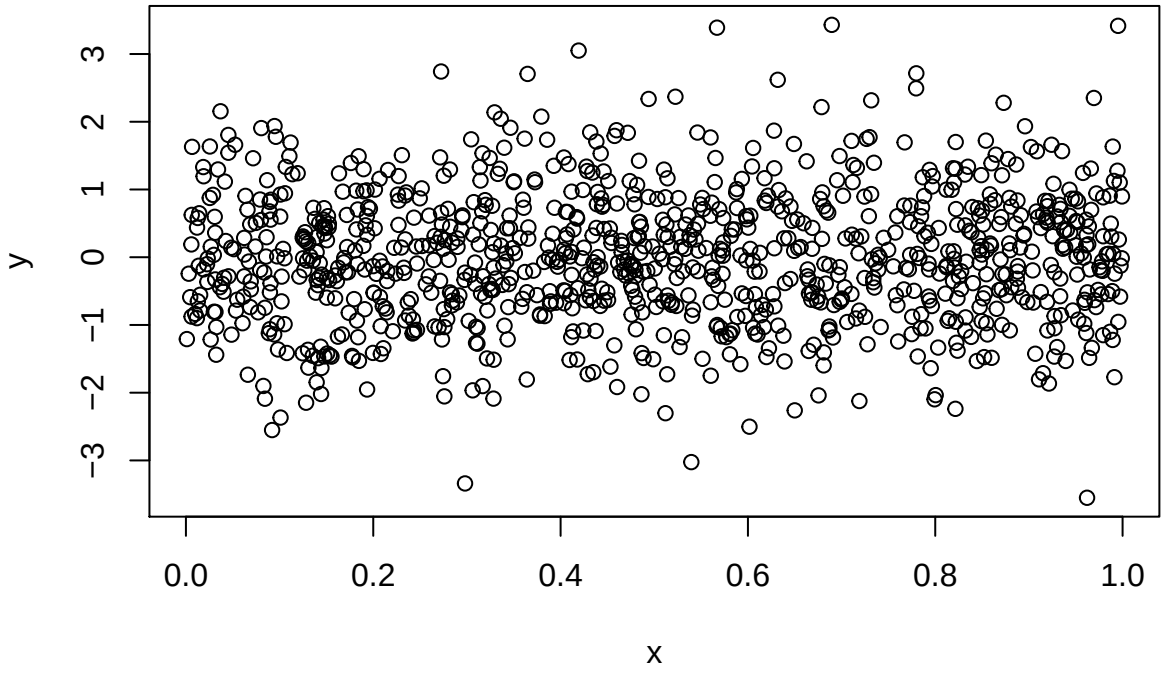


The most versatile of the plotting functions, however, is the *plot()* function. Depending on how many arguments are given to *plot()*, it will draw a line plot, a scatter plot connected by lines, or a scatter plot with markers.

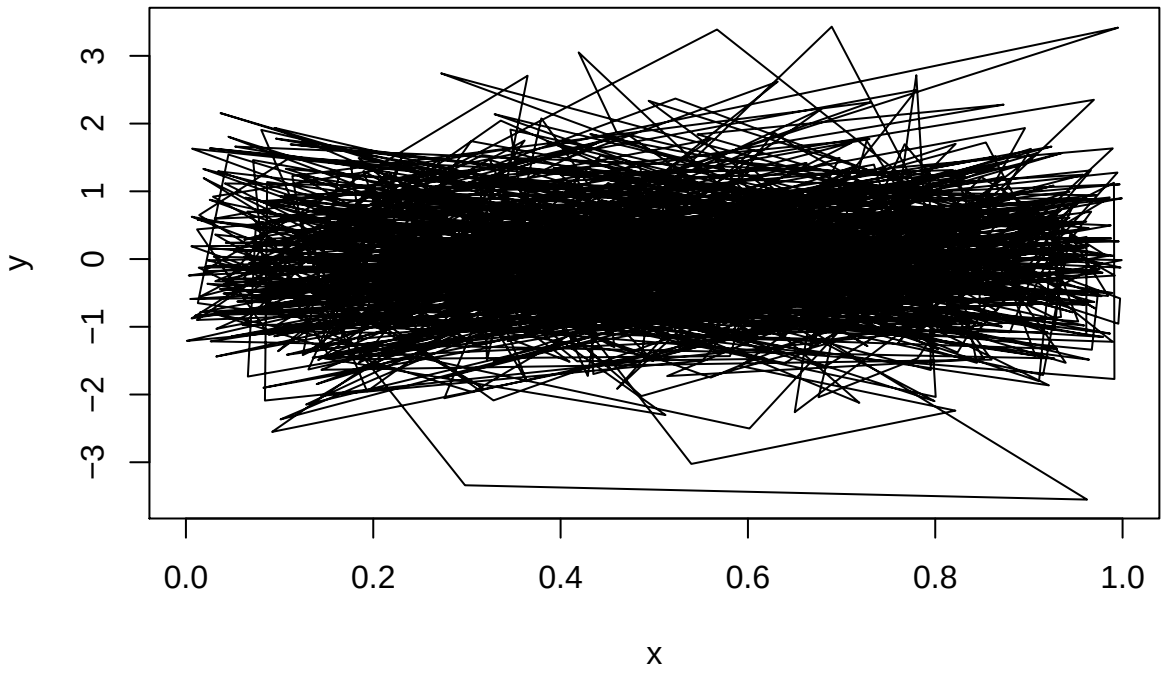
```
x <- runif(1000)
y <- rnorm(1000)
plot(x)
```




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

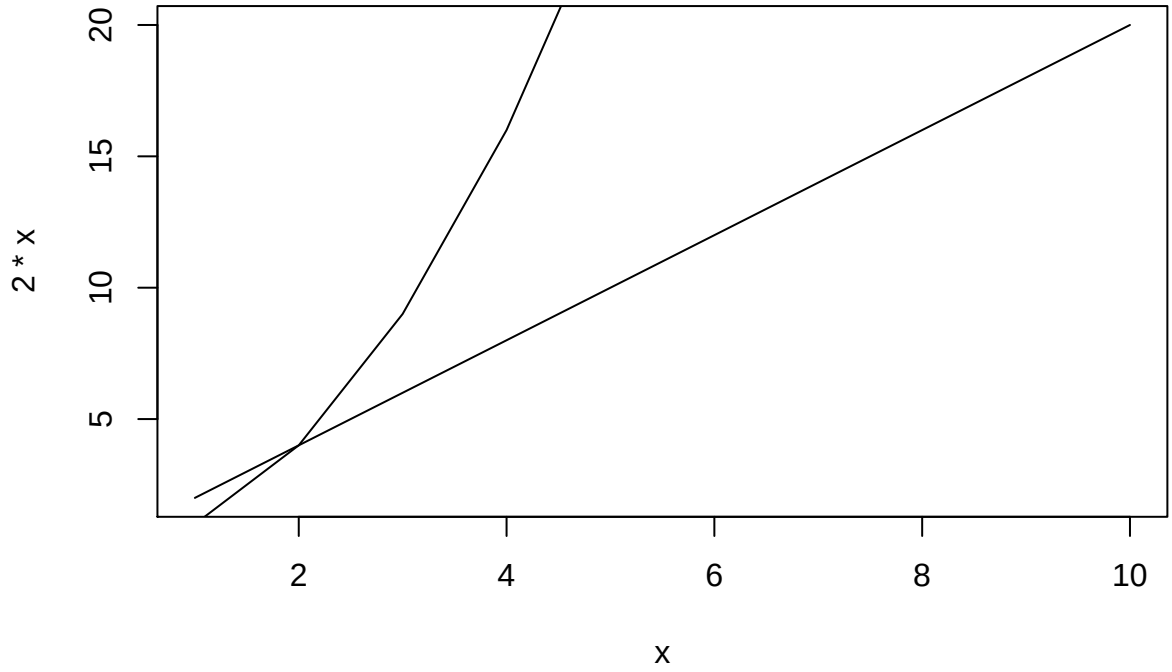



```
plot(x, y, "l")
```



If you would like more fine-grained control over your plotting, you can also call functions that do fewer steps. For example, the *lines()* and *points()* functions only plot the data without making a new window. This lets you plot multiple times to the same window!

```
x <- seq(1, 10)
plot(x, 2 * x, 'l')
lines(x, x ^ 2)
```



All of these functions allow you ot make much more beautiful plots, including adding axis labels, titles, and change the color and size of all components of the plot. We'll review how to use these functions in the next section.

Exercises

1. Make a bar plot showing increasing values from 5 to 12.
2. Make histograms showing a normal distribution, a uniform distribution, and a gamma distribution.
3. As you get more data, the effect of noise is decreased, allowing you to investigate it in more detail. In the case of representing the distribution of the data, how many histogram "breaks (i.e. bins) can be made to make a nice plot of 100 data points? 300 data points? 1000 data points? 10000?
4. Plot the squares of the sequence from 0 to 10 with steps of 0.01.
5. Plot the same sequence, this time adding noise to the previous plot.
6. Plot the "true" values as a line on top of the previous plot.