Exploring Distributions in R

UIUC Math Camp 2020

Miles D. Williams

In this Math Camp session, we're going to explore working with distributions in `R`. Things we'll cover include:

1. What is a probability distribution?
2. Continuous vs. discrete distribution functions.
3. Theoretical vs. empirical distributions in `R`.
4. Tools to summarize and visualize distributions in `R`.

Let's get started!

```r
# libraries
library(tidyverse) # for grammar
library(ggridges)  # I like the plotting theme for ggridges
```

What is a probability distribution?

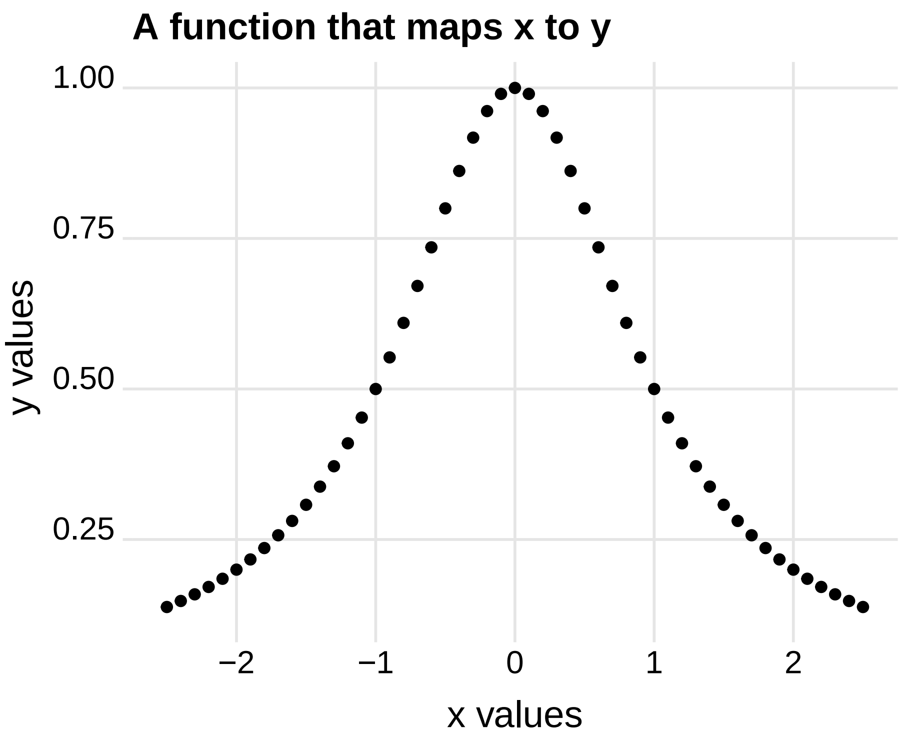
A *probability distribution* or *density* is a mathematical function that describes the probability of observing an occurrence of some outcome. As you learned in the session on *sets, intervals, and functions*, a function is a mapping between some n-dimensional input and some n-dimensional output. A probability function is a particular type of function that takes as input some real valued number or integer, and returns a value between 0 and 1. Formally, a probability distribution is a function such that

$$f : \mathbb{R}^n \to [0, 1].$$

As an example, say we have the function $f : x \to y$ where $x \in \mathbb{R}$ and $y \in (0, 1]$. In words, f is a function that takes as input any value x which lives in the domain of real numbers, and maps this to a value y that lives in the domain of real numbers between 0 and 1, with 0 non-inclusive.[1] Such a function could look like this

$$y = f(x) = \frac{1}{1+x^2}.$$

Now, say we have a *vector* of x values where $x = \{-2.5, -2.4, -2.3, ..., n-0.1, n, n+0.1, ..., 2.3, 2.4, 2.5\}$. We can use our function f to map these x values to corresponding y values on an x-y plot:



Looking at the correspondence between x and y values with a plot like this helps to demonstrate the intuition behind "mapping." In the above figure, our function f literally helps us to *map* out x-y coordinates given values of x.

With this particular example, we can think of our function f that we've just defined as a type of probability density function since it takes a real number and converts it into a probability—a value between 0 and 1 that describes the proportion of the time that we can expect to observe a particular value of x, or perhaps some outcome given a value of x.

In short, the function f is a formal way of characterizing the distribution of values for

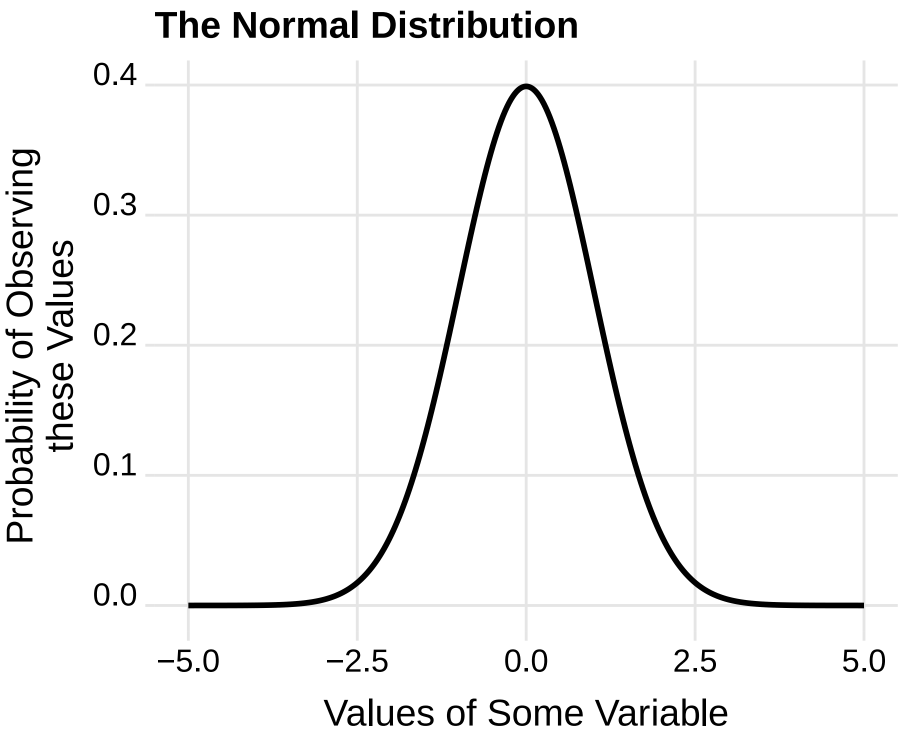
[1]The "0 non-inclusive" part here isn't strictly a requirement for a probability function—it's just a feature of the particular function I've specified here.

some variable, in this case x.

An actual probability density function

Now, in practice a researcher doesn't just use any odd probability density function they can conjure up (as we just did above). There are countless classes and types of distribution functions, each a *theoretical* model characterizing the distribution of certain types of variables.

One of the most common (and probably one that you were most likely to have been exposed to prior to now) is the normal or Gaussian distribution. This usually looks like this:

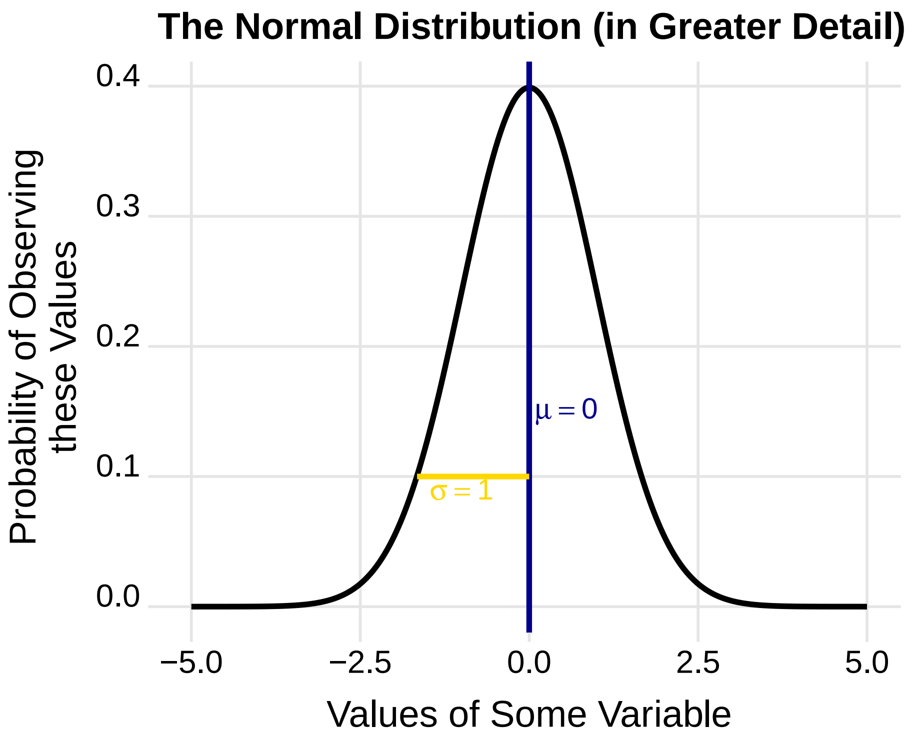


The theoretical probability density function (p.d.f.) that describes the normal distribution is,

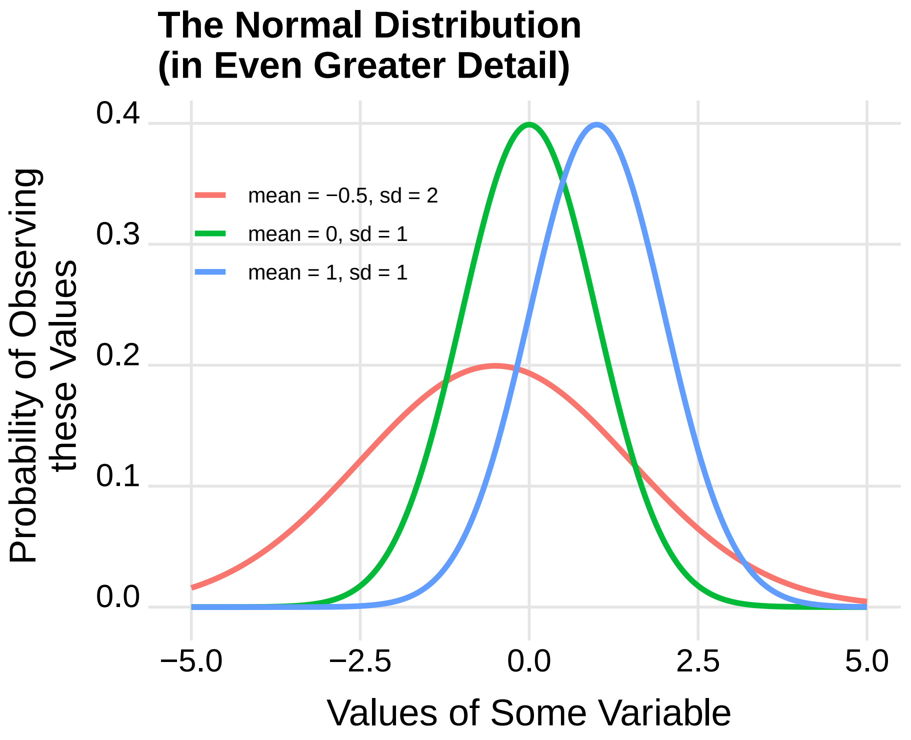
$$f(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

In addition to mapping values of the variable x to unique probabilities, the p.d.f. also has two *parameters* that provide information about the variable's *mean* (μ) and *standard deviation* (σ).

As shown below, the value of μ determines the *central tendency* of the variable's distribution, while σ determines the variable's *dispersion* around the mean.



Larger values of σ will mean a wider curve describing the distribution of a variable, while differing values of μ will change where the curve reaches its peak—as you can see below.



Continuous vs. discrete distributions

The normal distribution describes the probability of observing values of some continuous variable. But in many applied settings researchers deal with *discrete* variables. For example, a political scientist might use a 7 point political ideology scale that only takes the values 0, 1, 2,. . . , 5, 6, to predict the likelihood of voting for a parliamentary candidate (with a vote for said candidate being coded as 1, and a vote for any other candidate as 0).

In these instances, instead of relying on a probability density function, we use a *probability mass function* (p.m.f.). This function tells us the probability that a variable takes precisely a given value. For the binary measure of whether a voter casts a ballot in favor of our hypothetical parliamentary candidate, the p.m.f. would be given as

$$\mathrm{Bern}(x) = \begin{cases} p, & \text{if } x = 1, \\ 1 - p, & \text{if } x = 0. \end{cases}$$

This is the equation for the *Bernoulli distribution*—another common probability distribution function.

A classic example of when this distribution applies is in describing the results of a coin toss.

The value p denotes the probability that the coin comes up Heads, while $1 - p$ denotes the probability that the coin comes up Tails.

The Bernoulli distribution also applies to votes cast for a particular parliamentary candidate. p describes the probability that an individual voter will vote for candidate A, while $1 - p$ describes the probability that this voter will vote for candidate B.

A related distribution is the *Binomial distribution*, which has the p.m.f.:

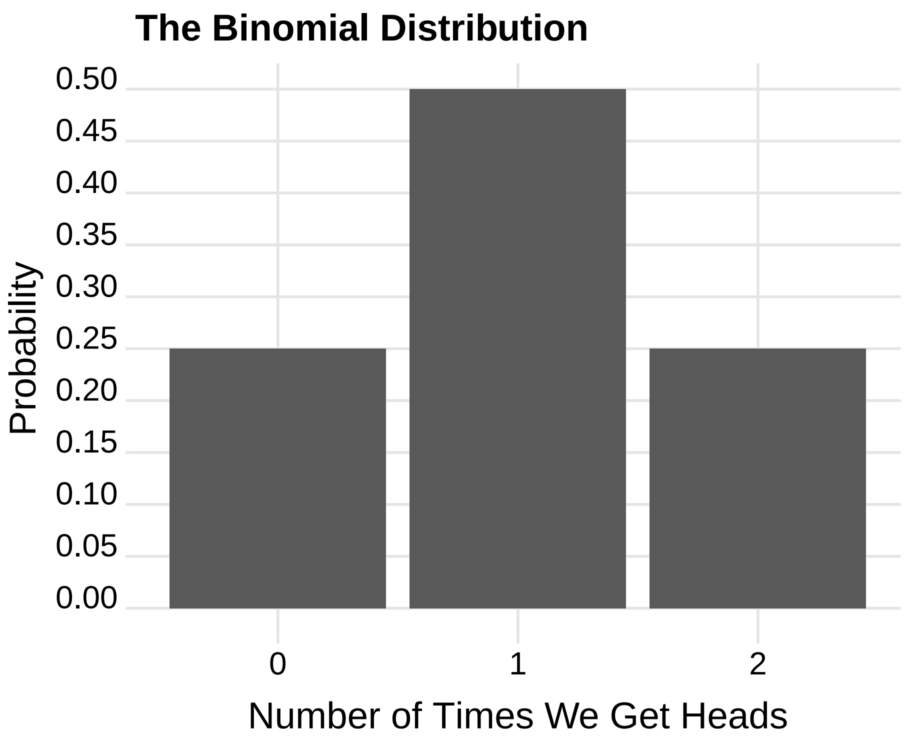
$$\mathrm{Bin}(x) = \frac{n!}{k!(n-k)!}(1-p)^{n-k}.$$

The Binomial distribution is actually a generalization of the Bernoulli distribution. The Binomial distribution describes the probability of k successes in n independent trials. The Bernoulli distribution describes the special case where $n = 1$—that is, there is only one trial, as in the toss of a coin.

A binomial distribution, on the other hand, is useful more generally for describing the probability mass of a discrete variable with more than one potential outcome. Say, for instance, we wanted to know the probability of getting a certain number of Heads after two coin tosses. We know that for a fair coin, the probability of Heads after one toss is $p = 1/2$.

But what about the probability of getting two heads after 2 tosses? As it turns out, this is calculated simply as $p \times p = p^2$. If we expect that for any one toss, we have Heads half the time, then half the time we have Heads on the first toss, half the time we can expect to have Heads on the second. Thus, we can expect to get two heads in a row one fourth of the time.

The figure below summarizes the p.m.f. for a two trial coin toss. As it shows, we can expect to get at least one Head in two tosses $p = 1/2$ proportion of the time. Meanwhile, the probability of either two Heads in a row, or of no Heads twice in a row is $p = 1/4$.



Theoretical versus empirical distributions

Up to now, we've been dealing with *theoretical* distributions. These are called "theoretical" because they are precisely that—they tell us, in theory, the proportion of the time, or frequency, by which we can expect to observe certain values of a given variable.

Say we have a random variable x, and say that x is best summarized as following a normal distribution. The formula for a normal distribution, recall, is

$$f(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

While this equation is a good approximation of x's distribution, it is not going to be strictly correct in summarizing the *empirical* distribution of x.

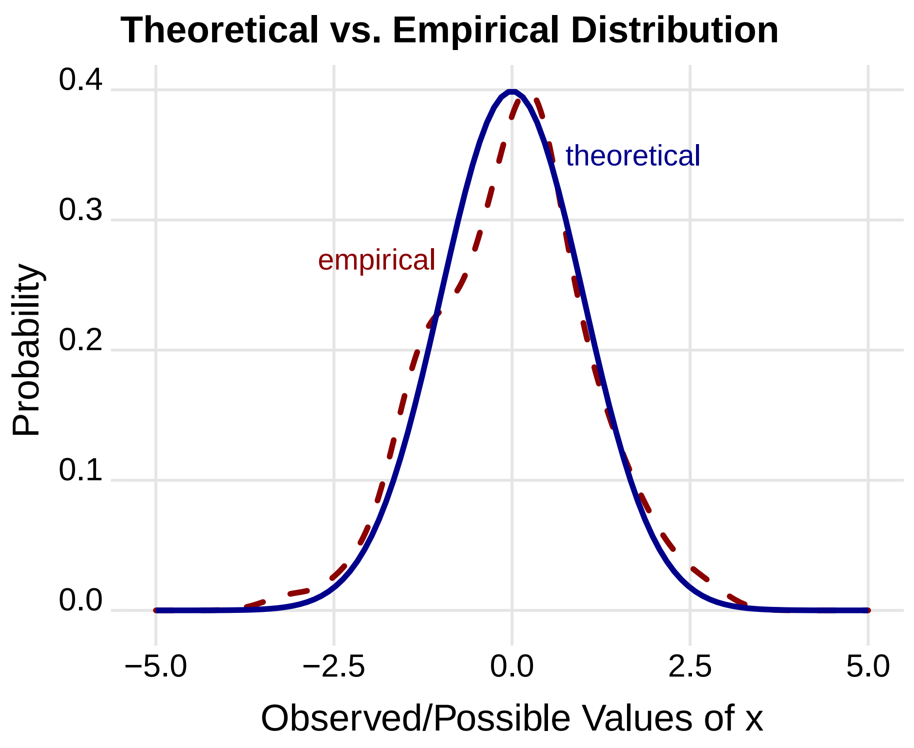
To demonstrate this point, let's simulate some empirical data in `R`. Say we have a random variable x where $x \sim \mathcal{N}(\mu = 0, \sigma = 1)$—this notation simply means that we're simulating random draws from a normal distribution with mean of 0 and standard deviation of 1. We simulate this in `R` as follows:

```r
set.seed(111) # setting the seed ensures I get the same output each time
x = rnorm(
  n = 100,
  mean = 0,
  sd = 1
)
head(x)
```

```
## [1]  0.2352207 -0.3307359 -0.3116238 -2.3023457 -0.1708760  0.1402782
```

I've just created a vector `x` with 100 observations or values, where each of these values was randomly drawn from a normal distribution—the function to do this in `R` is `rnorm()`. Above, I used the `head()` function to just show the first 6 values of `x`.

Let's take a look at the empirical distribution of values in `x`, and compare this to the theoretical distribution from which these values were drawn:



The empirical distribution as summarized above is calculated using what's called a *kernel density estimate*. This is just a smoothed out version of a *histogram*, which is another useful way for summarizing the empirical distribution of a variable. The kernel density for `x` just gives us the best approximation of the probability density of the variable based on the observed values of the variable.

Clearly, the computed empirical distribution is not a perfect match for the theoretical distribution. It's a bit skewed, and a little lumpy.

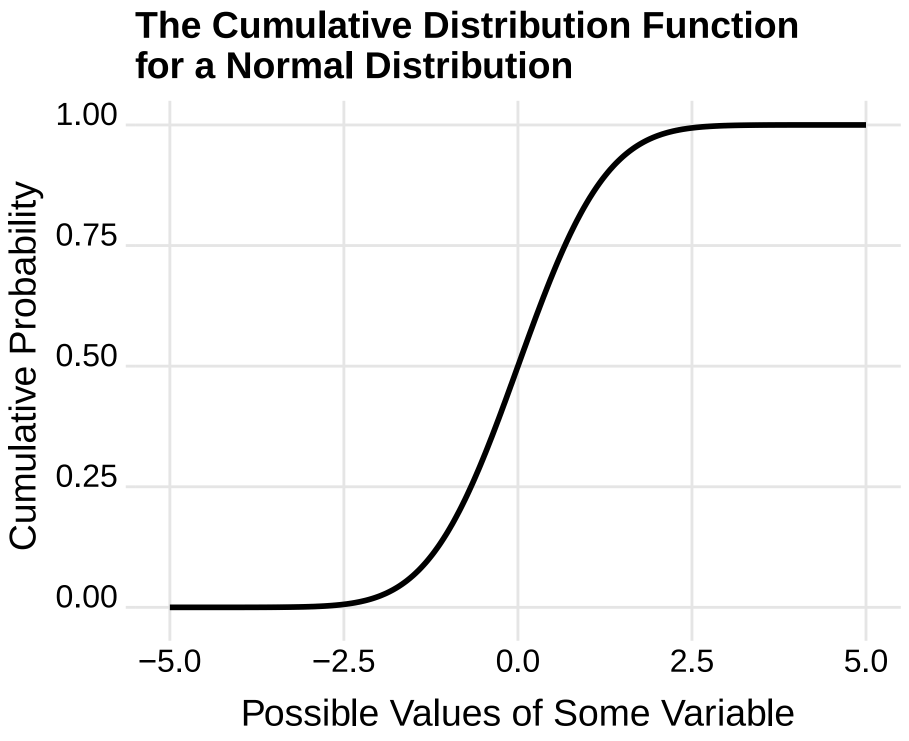
This skeweness and lumpiness is common for empirical distributions—especially in the context of finite or small samples. However, the more observations of a random variable we have, the more the empirical distribution begins to look like the theoretical distribution (this is the *central limit theorem*).

Cumulative distribution functions

In addition to using p.d.f.'s and p.m.f.'s, researchers also commonly use what are called *cumulative distribution functions* (c.d.f.). In the case of a continuous variable, the c.d.f. tells us the cumulative area under the curve of a p.d.f.

In other words, the c.d.f. is the *integral* of the p.d.f.

For the normal distribution, the c.d.f. looks like this:



The equation for the c.d.f. of the normal distribution is given as

$$\int_a^b f(x|\mu,\sigma)dx.$$

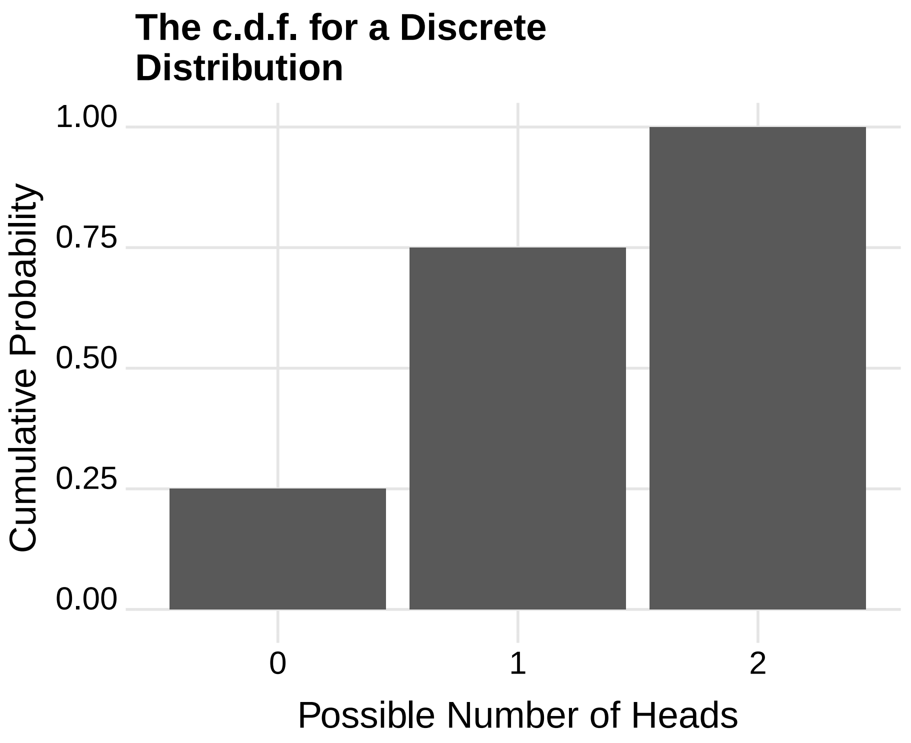
This is equivalent to the probability of observing any value of x in the range of values from a to b where $a, b \in x$ and $a < b$. Formally, this is given as

$$\Pr(a < x \leq b).$$

For a discrete variable, rather than integrating over the p.d.f., we simply take the sum over the p.m.f. For the Binomial distribution, this is

$$\sum_{x_i \leq x} \text{Bin}(x_i)$$

For our coin toss example where we wanted to know the probability of getting a certain number of heads after two tosses, the c.d.f. looks like this:



Tools in `R` for summarizing and plotting distributions

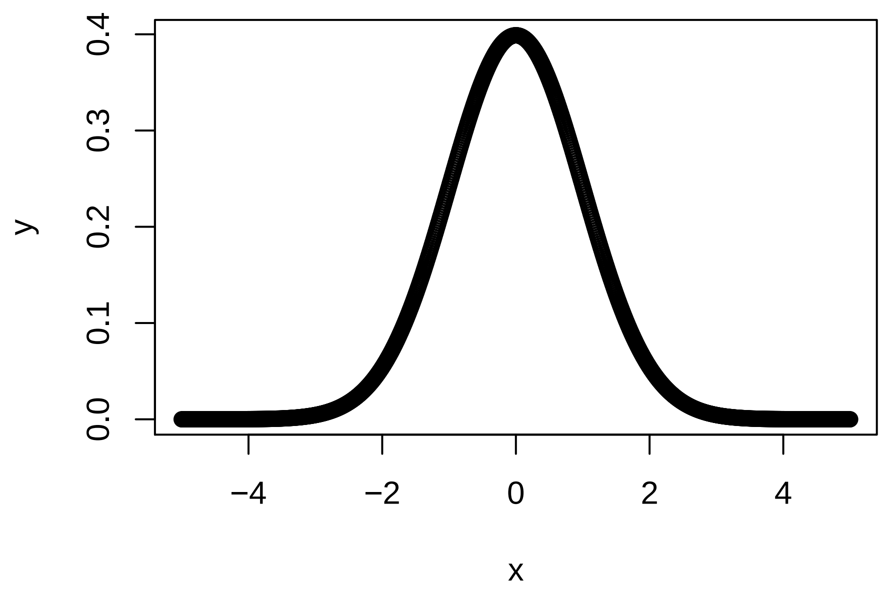
`R` provides a buffet of tools for summarizing and plotting all kinds of distributions. To keep things simple, let's just focus on functions related to the normal distribution.

The following are base `R` functions associated with the normal distribution:

- `dnorm`: returns the theoretical probability density of a normal distribution given mean and standard deviation over possible values of a variable.
- `pnorm`: returns the cumulative probability of a normal distribution given mean and standard deviation over possible values of a variable.
- `qnorm`: returns the value of a variable given probabilities from a c.d.f.
- `rnorm`: generates random draws from a normal distribution.

Here's some example code for using `dnorm` to generate probabilities from a normal distribution:

```r
# 1. Create a vector 'x' with values that range from -5 to 5 in
#    increments of 0.01.

x = seq(from=-5,to=5,by=0.01)

# 2. Get vector 'y' which is the corresponding probabilities
#    for values in 'x'. These are returned by a call to `dnorm`

y = dnorm(x=x) # use defaults for mean and s.d.

# 3. Plot y over values of x.

plot(x,y)
```

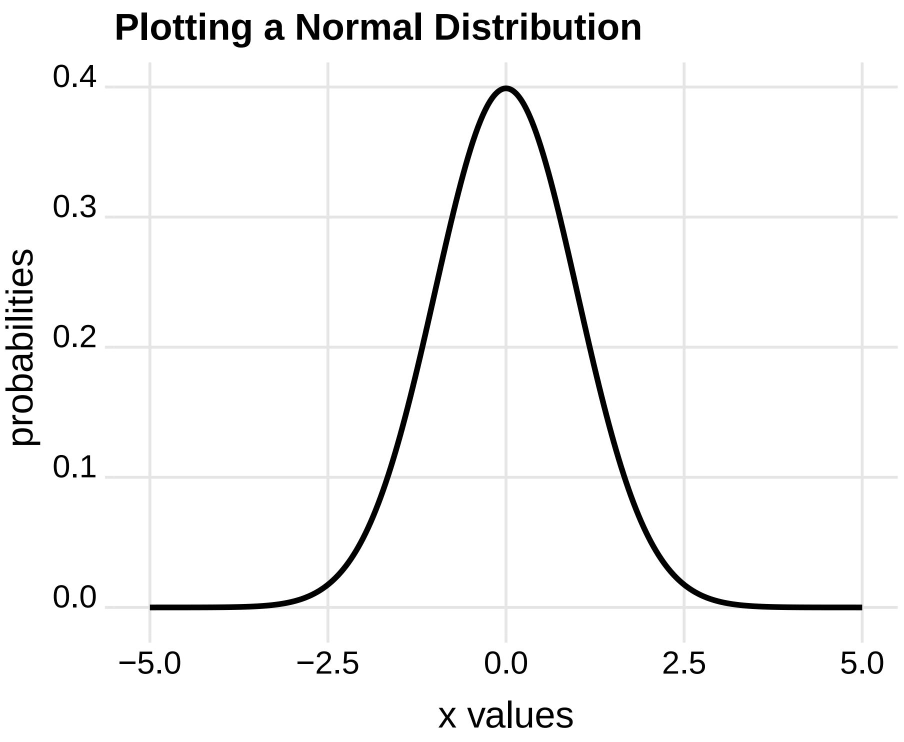


The above plot is produced with the base `R` plotting function, `plot()`. By default, this function returns a point plot and names the x-y axes according to the names of the variables used as inputs.

Though the syntax can be a little more involved, `ggplot()` in the `ggplot2` package offers a great deal more flexibility for making pretty-to-look-at graphs:

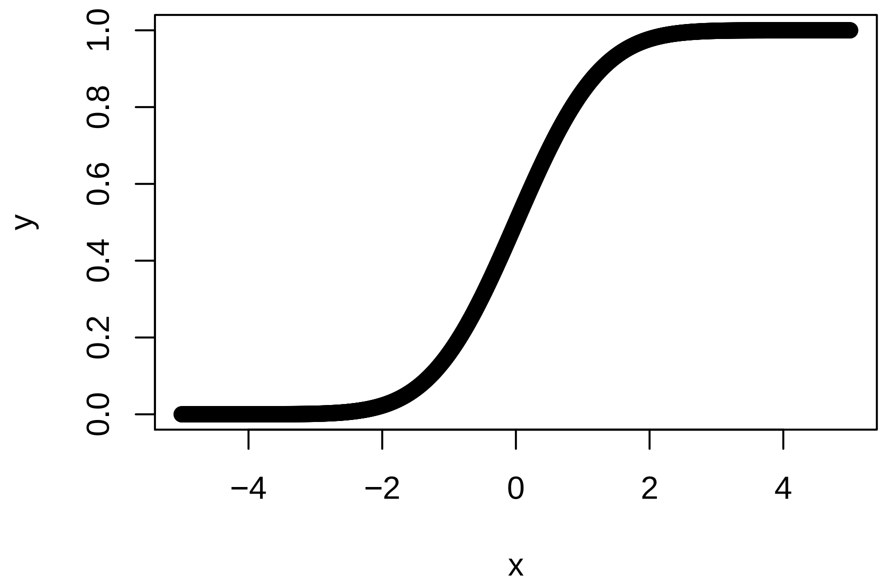
```
ggplot(data=NULL) + # make empty plot (data=NULL by default)
  aes(
    x = x,           # after using '+' to signify that you're
    y = y            # adding a new graphical layer, use aes()
  ) +                # to include aesthetics (x and y values to plot)
  geom_line(
    size = 1         # then tell ggplot that you want to use a
  ) +                # line of size '1' in plotting x-y values
  labs(
    x = 'x values', # add x-y labels and give the plot a title
    y = 'probabilities',
    title = 'Plotting a Normal Distribution'
  ) +
```

```r
  theme_ridges(      # use the ridges theme from the ggridges package
    center_axis_labels = T
  )
```



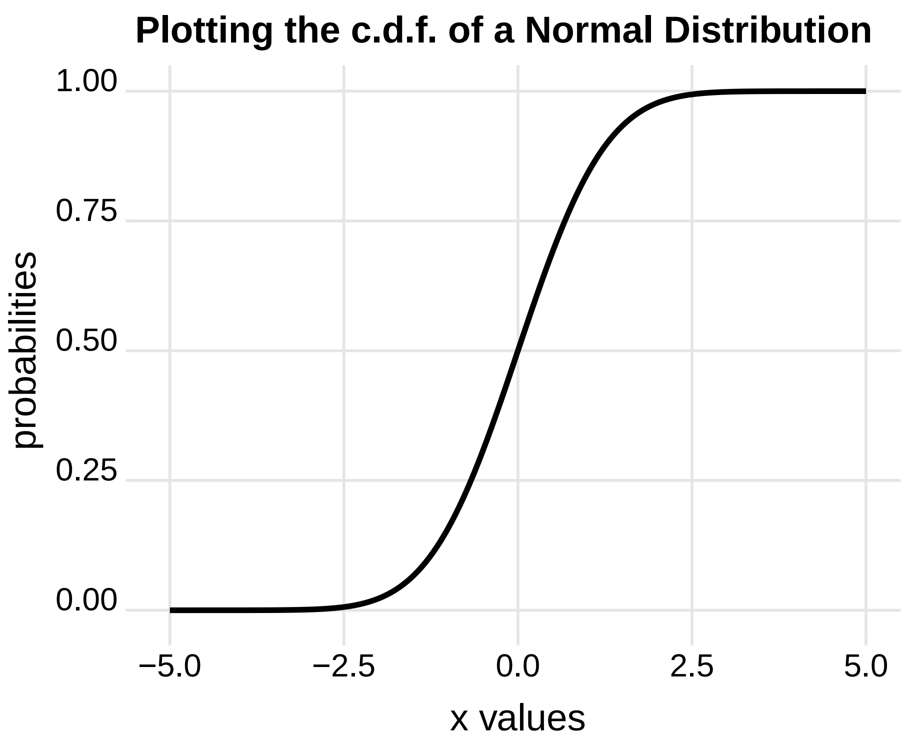
We can similarly use `pnorm` to plot the c.d.f. of the normal distribution:

```r
# 1. Create a vector 'x' with values that range from -5 to 5 in
#    increments of 0.01.

x = seq(from=-5,to=5,by=0.01)

# 2. Get vector 'y' which is the corresponding probabilities
#    for values in 'x'. These are returned by a call to `pnorm`

y = pnorm(x) # use defaults for mean and s.d.

# 3. Plot y over values of x.

plot(x,y)
```



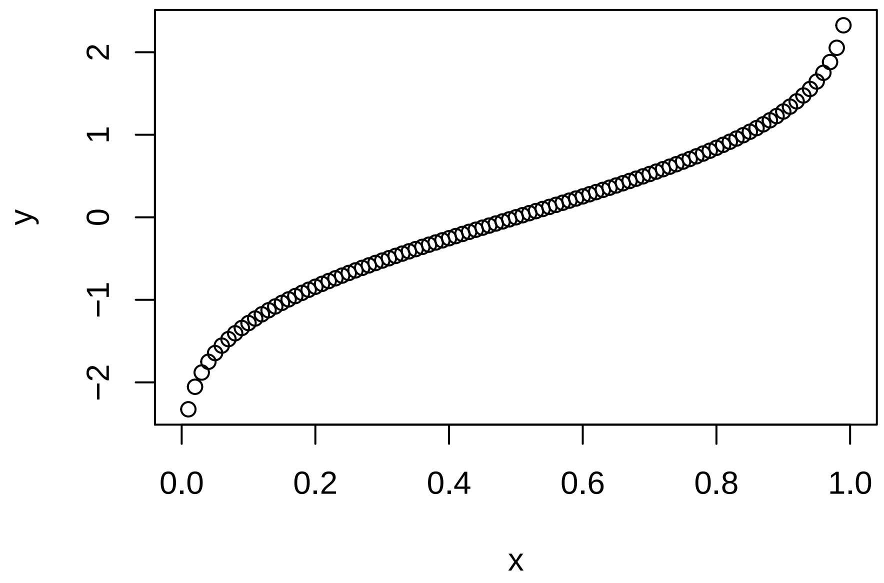
Again, we can use `ggplot` to add a little more flare to the visualization:

```
ggplot(data=NULL) + # make empty plot (data=NULL by default)
  aes(
    x = x,          # after using '+' to signify that you're
    y = y           # adding a new graphical layer, use aes()
  ) +               # to include aesthetics (x and y values to plot)
  geom_line(
    size = 1        # then tell ggplot that you want to use a
  ) +               # line of size '1' in plotting x-y values
  labs(
    x = 'x values', # add x-y labels and give the plot a title
    y = 'probabilities',
    title = 'Plotting the c.d.f. of a Normal Distribution'
  ) +
  theme_ridges(
    center_axis_labels = T
  )
```



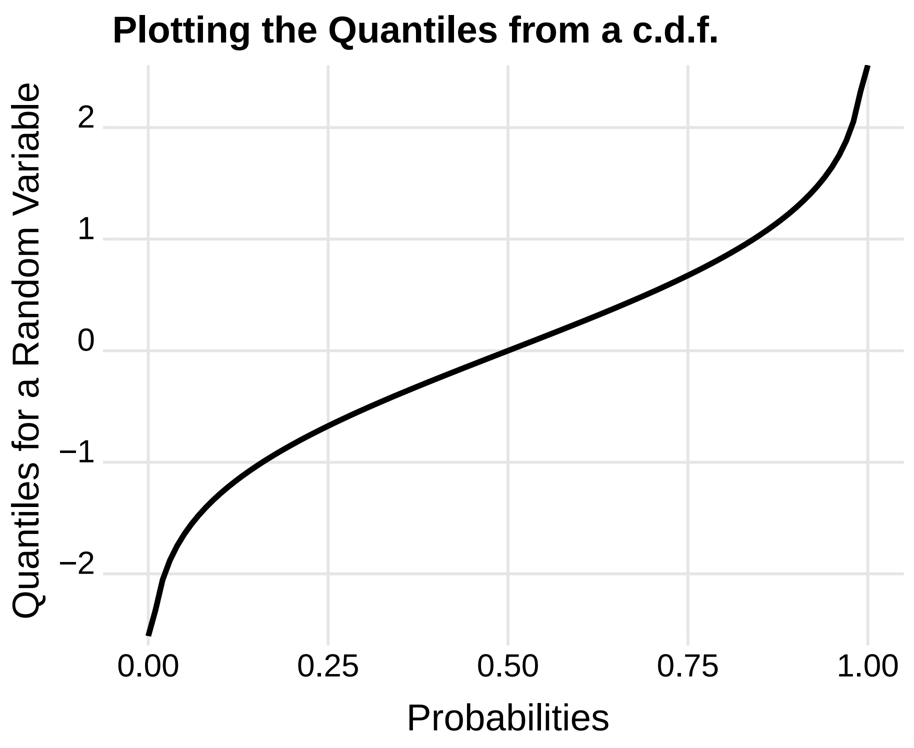
While `pnorm` returns probabilities, `qnorm` returns quantiles based on cumulative probabilities:

```r
# 1. Create a vector 'x' of probabilities from 0 to 1.

x = seq(from=0,to=1,by=0.01)

# 2. Get vector 'y' which is the corresponding quantiles
#    for values in 'x'. These are returned by a call to `qnorm`

y = qnorm(x) # use defaults for mean and s.d.

# 3. Plot y over values of x.

plot(x,y)
```



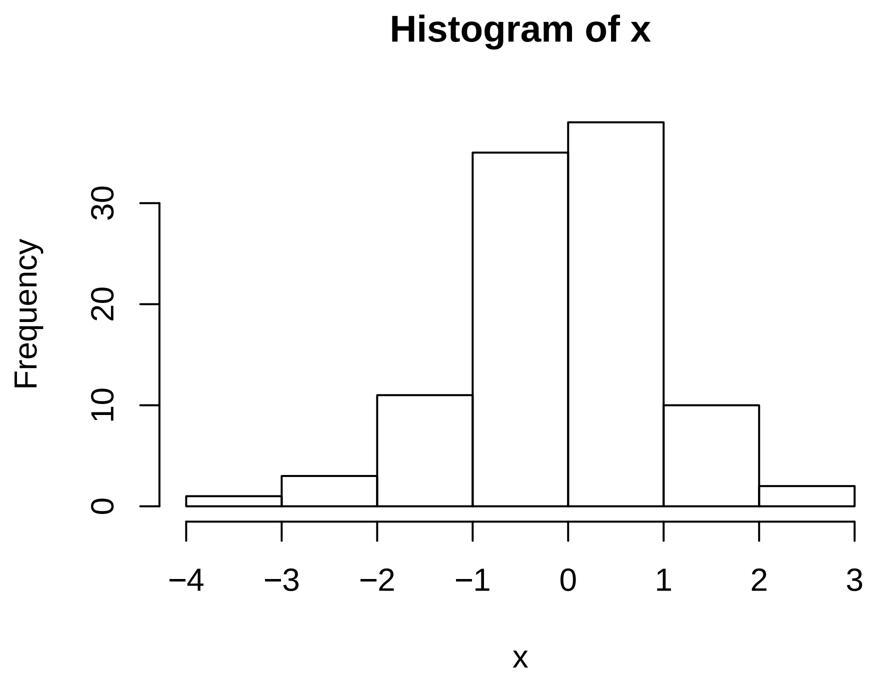
And once again with ggplot:

```
ggplot(data=NULL) + # make empty plot (data=NULL by default)
  aes(
    x = x,           # after using '+' to signify that you're
    y = y            # adding a new graphical layer, use aes()
  ) +                # to include aesthetics (x and y values to plot)
  geom_line(
    size = 1         # then tell ggplot that you want to use a
  ) +                # line of size '1' in plotting x-y values
  labs(
    x = 'Probabilities', # add x-y labels and give the plot a title
    y = 'Quantiles for a Random Variable',
    title = 'Plotting the Quantiles from a c.d.f.'
  ) +
  theme_ridges(
    center_axis_labels = T
  )
```



Finally, we can use `rnorm` to simulate empirical values from a normal distribution:

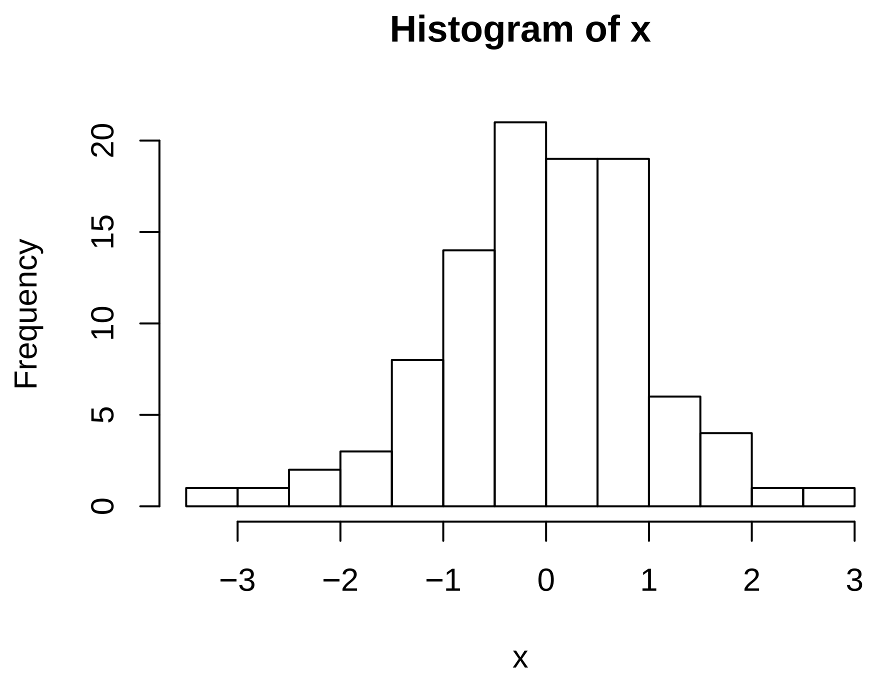
```
# 1. Get vector of random draws from a normal distribution.

x = rnorm(n=100) # use defaults for mean and s.d., and n = 100

# 2. Plot a histogram of `x`

hist(x)
```



`hist` is another base `R` plotting function that returns a histogram. This is just a way of summarizing the frequency at which values of a random variable occur within a particular “bin” or range of values in the data. In the above example, `hist` by default selected 7 equally sized bins where each bar signifies the number of observations in `x` that have values that fall within said bin.

We can override the number of bins if we so choose:

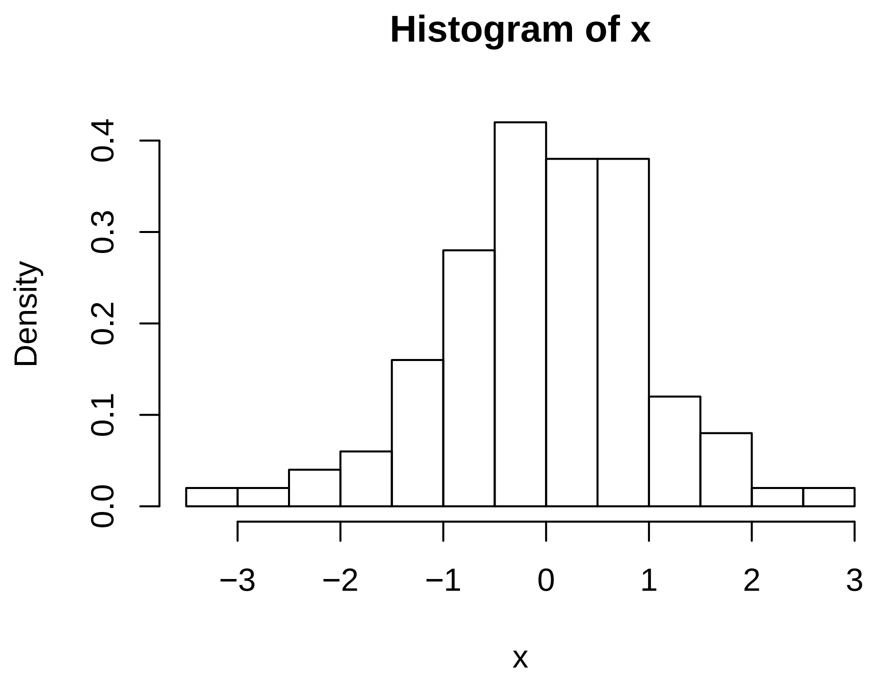
```r
hist(x,breaks = 20)
```



In the above, I used `breaks = 20` to indicate that I wanted to summarize the data by 20 bins of equal size.

We can also change whether `hist` reports back frequencies or probabilities:

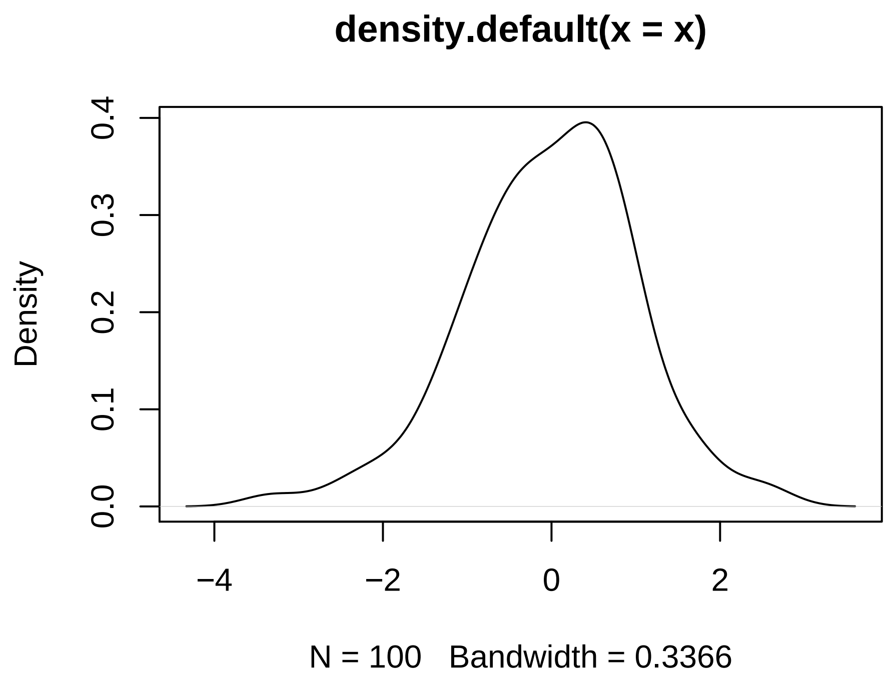
```
hist(x,breaks=20,freq=F)
```



By setting `freq=F`, I told `R` I wanted a histogram that reports probabilities rather than frequencies—or more literally, I told `R` that "frequencies is **false**".

Before, I highlighted that we can also use kernel density estimates to summarize the empirical distribution of a continuous variable. Using base `R` commands, this is done like so:

```r
plot(density(x))
```



I used the `density` command to return a set of x-y coordinates that map values of `x` to a set of probabilities using a kernel density function. This is what the output of `density` looks like:

```
density(x)
```

```
##
## Call:
##  density.default(x = x)
##
## Data: x (100 obs.);  Bandwidth 'bw' = 0.3366
##
##        x                 y
##  Min.   :-4.3331   Min.   :0.0001336
##  1st Qu.:-2.3493   1st Qu.:0.0138308
##  Median :-0.3656   Median :0.0521889
##  Mean   :-0.3656   Mean   :0.1258972
##  3rd Qu.: 1.6182   3rd Qu.:0.2430554
##  Max.   : 3.6020   Max.   :0.3955056
```

I can also directly pull `x` and `y` values from a `density` object:

```r
d = density(x)
head(d$x) # x values
```

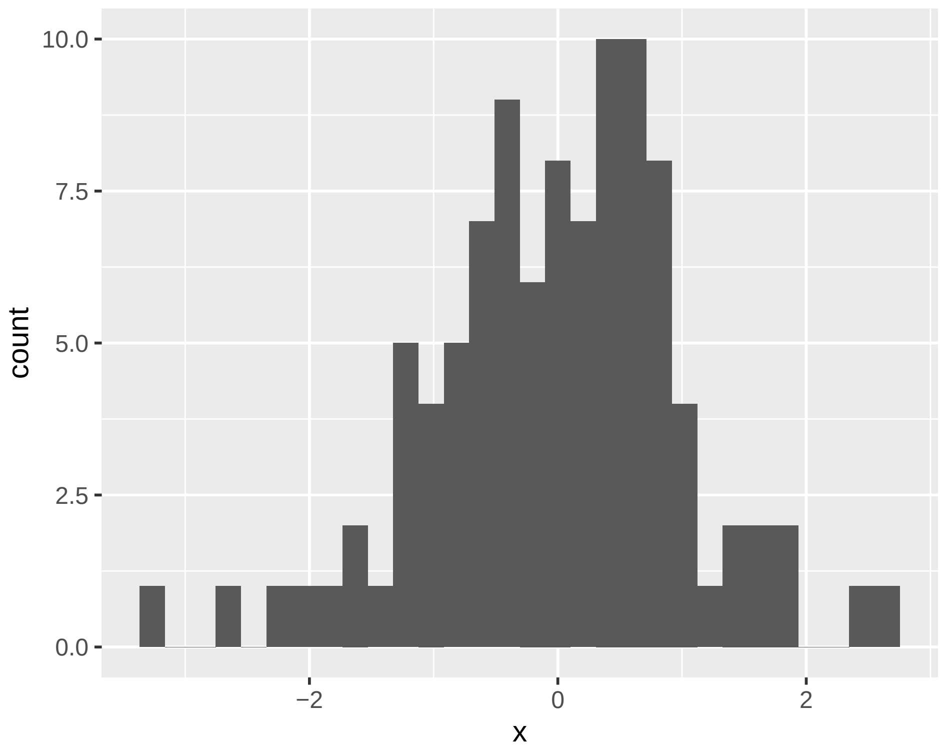
```
## [1] -4.333063 -4.317535 -4.302006 -4.286478 -4.270950 -4.255421
```

```r
head(d$y) # y values
```

```
## [1] 0.0001336469 0.0001533848 0.0001753343 0.0001997857 0.0002281795
## [6] 0.0002596132
```

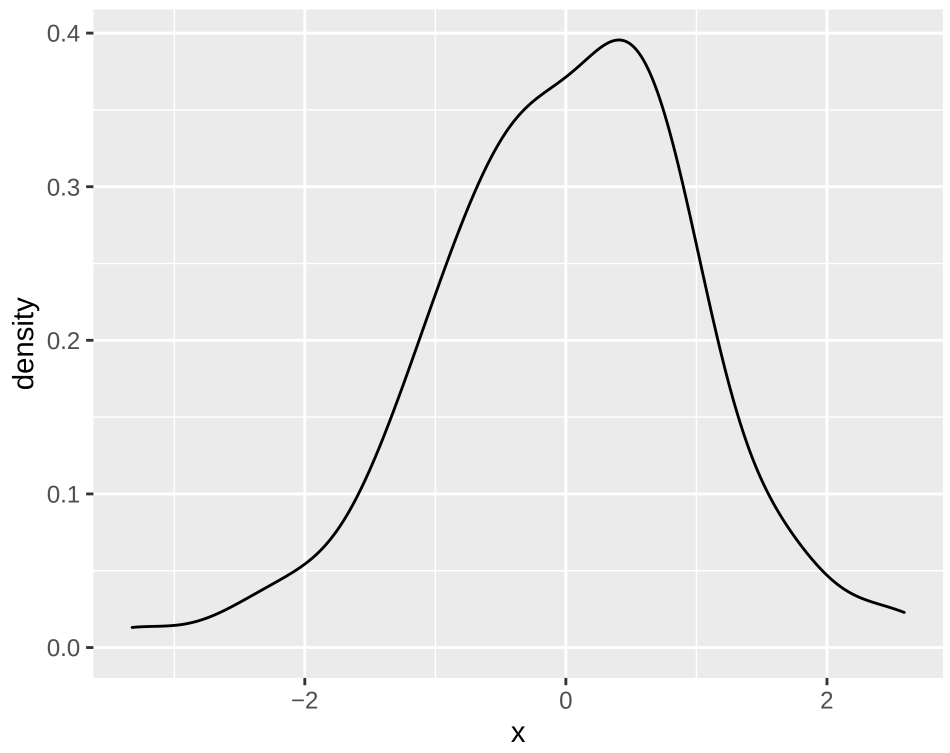
We can also generate similar plots with `ggplot`. Here's how to plot a histogram:

```r
ggplot() +
  aes(x) +
  geom_histogram()
```



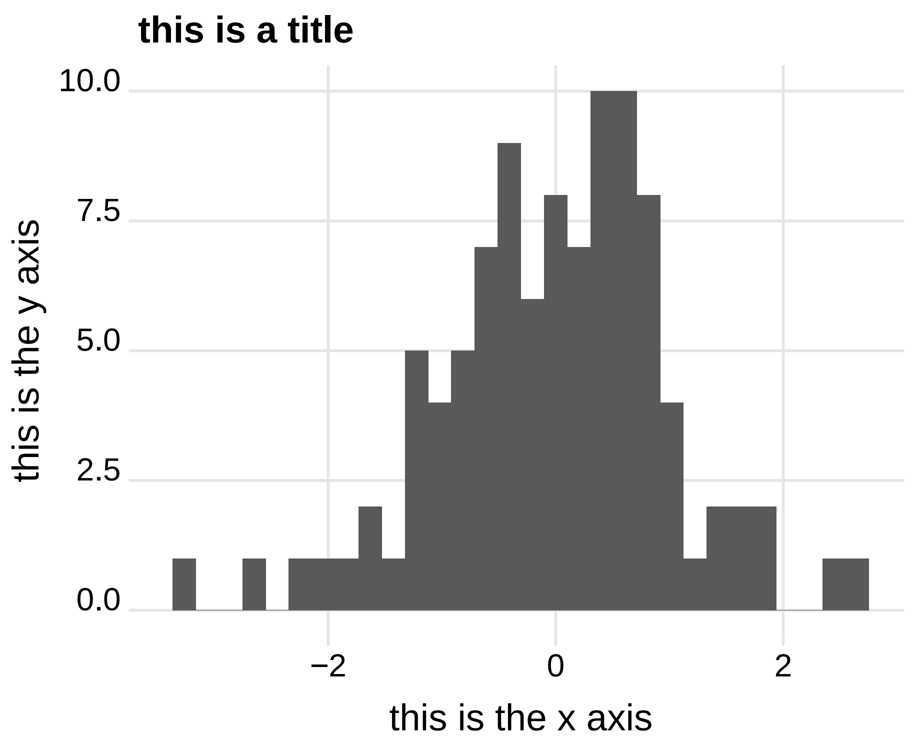
And here's how to plot densities:

```r
ggplot() +
  aes(x) +
  geom_density()
```



As before, we can do all sorts of things to spruce these figures up.

```
ggplot() +
  aes(x) +
  geom_histogram() +
  labs(
    x = 'this is the x axis',
    y = 'this is the y axis',
    title = 'this is a title'
  ) +
  theme_ridges(
    center_axis_labels = T
  )
```



Outside of visualizations, `R` provides a number of useful functions for summarizing data that will print out results in your console. For example, you can use the `summary` function to return all sorts of useful info about a given variable:

```
summary(x)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -3.323335 -0.618477  0.005538 -0.016867  0.640291  2.592227
```

The above shows the minimum, maximum, median, mean, and first and third quartiles for `x`.

Say you're working with what's called a "data frame" in `R` and you want summaries for all the variables in the data frame at once. You can use `summary` for this, too:

```
# create a tidy data frame with "tibble" from the tidyr package
df = tibble(
  x = rnorm(n=100),
  y = rnorm(n=100)
)
summary(df)
```

```
##        x                   y
##  Min.   :-2.16141   Min.   :-2.29131
##  1st Qu.:-0.55068   1st Qu.:-0.56589
##  Median : 0.08084   Median : 0.13311
##  Mean   : 0.13638   Mean   : 0.02312
##  3rd Qu.: 0.74866   3rd Qu.: 0.56042
##  Max.   : 2.62117   Max.   : 2.92603
```

You can also get the mean and standard deviation for a random variable:

```
mean(df$x) # mean of x
```

```
## [1] 0.136383
```

```
sd(df$x)   # s.d. of x
```

```
## [1] 0.9168738
```

If you want to get these for each of the variables in `df`, you can use the tools in the `tidyverse` of `R` packages:

```
df %>%
  summarize(
    mean.x = mean(x),
    mean.y = mean(y),
    sd.x = sd(x),
    sd.y = sd(y)
  )
```

```
## # A tibble: 1 x 4
##   mean.x mean.y  sd.x  sd.y
##    <dbl>  <dbl> <dbl> <dbl>
## 1  0.136 0.0231 0.917 0.958
```

Now it's your turn!

Now that you've had some opportunity to get acquainted with distributions and with some tools for summarizing and visualizing them in `R`, it's time to try some stuff out yourself.

You can download the .Rmd file that contains the problem set for this lesson from here.
You can see the questions in pdf format here.