# Additional Features of ggplot2

W. Evan Johnson, Ph.D.
Professor, Division of Infectious Disease
Director, Center for Data Science
Rutgers University – New Jersey Medical School

11/16/2023

# Case study: Describing Student Heights

Pretend that we have to describe the heights of our classmates to ET, an extraterrestrial that has never seen humans. We collect data on a set of humans and save it in the `heights` data frame:

```
library(tidyverse)
library(dslabs)
data(heights)
```

# Distribution Function

One way to convey the heights to ET is to simply send him this list of 1050 heights. But there are much more effective ways to convey this information, and understanding the concept of a distribution will help. To simplify the explanation, we first focus on male heights.

# Distribution Function

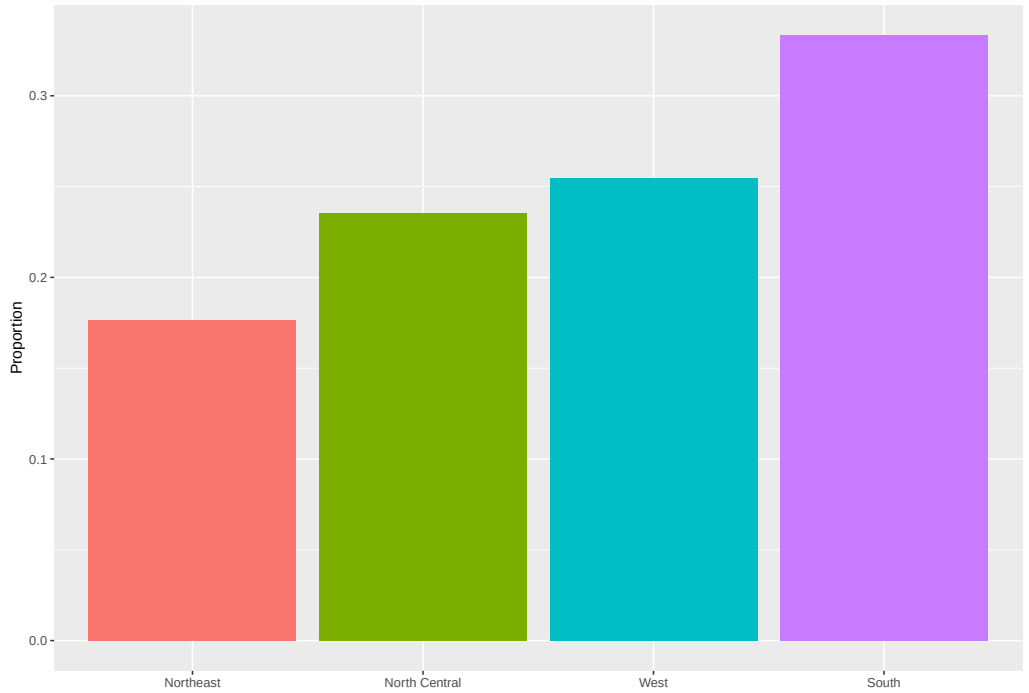
The most basic statistical summary of a list of objects or numbers is its distribution. The simplest way to think of a distribution is as a compact description of a list with many entries. For example, with categorical data, the distribution simply describes the proportion of each unique category. The sex represented in the heights dataset is:

```
##
##    Female      Male
## 0.2266667 0.7733333
```

This two-category **frequency table** is the simplest form of a distribution. We don't really need to visualize it since one number describes everything we need to know: 23% are females and the rest are males.

# Distribution Function

When there are more categories, then a simple barplot describes the distribution. Here is an example with US state regions:




This particular plot, a **barplot** simply shows us four numbers, one for each category. W

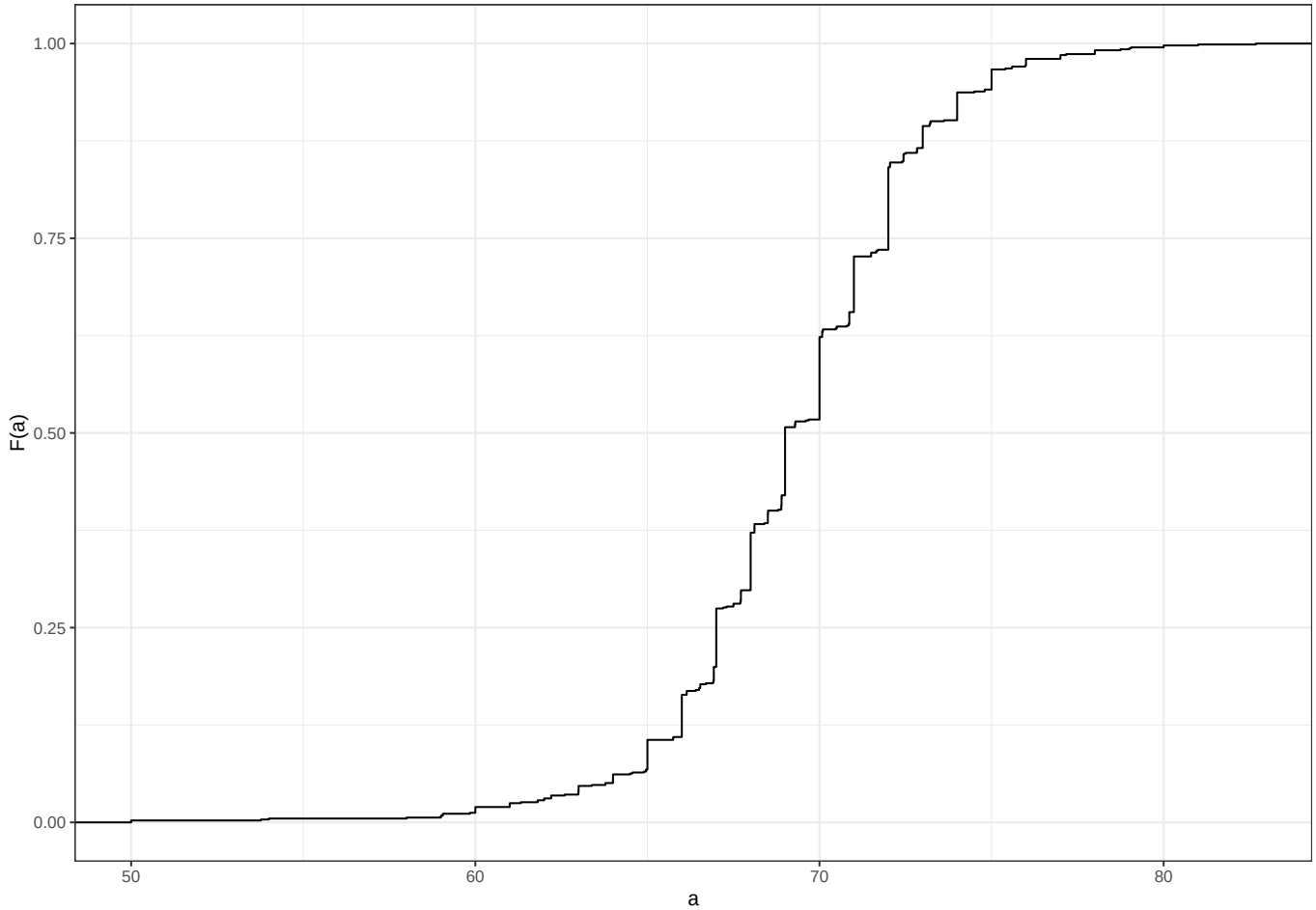
# Cumulative Distribution Functions

Statistics textbooks teach us that a more useful way to define a distribution for numeric data is to define a function that reports the proportion of the data below $a$ for all possible values of $a$. This function is called the cumulative distribution function (CDF). In statistics, the following notation is used:

$$F(a) = \Pr(x \leq a)$$

# Cumulative Distribution Functions

Here is a plot of $F$ for the male height data:



# Cumulative Distribution Functions

Similar to what the frequency table does for categorical data, the CDF defines the distribution for numerical data.

From the plot, we can see that 16% of the values are below 65, since $F(66) = 0.1637931$, or that 84% of the values are below 72, since $F(72) = 0.841133$, and so on. In fact, we can report the proportion of values between any two heights, say $a$ and $b$, by computing $F(b) - F(a)$.

# Cumulative Distribution Functions

This means that if we send this plot above to ET, he will have all the information needed to reconstruct the entire list. Paraphrasing the expression "a picture is worth a thousand words", in this case, a picture is as informative as 812 numbers.

A final note: because CDFs can be defined mathematically the word **empirical** is added to make the distinction when data is used. We therefore use the term empirical CDF (eCDF).

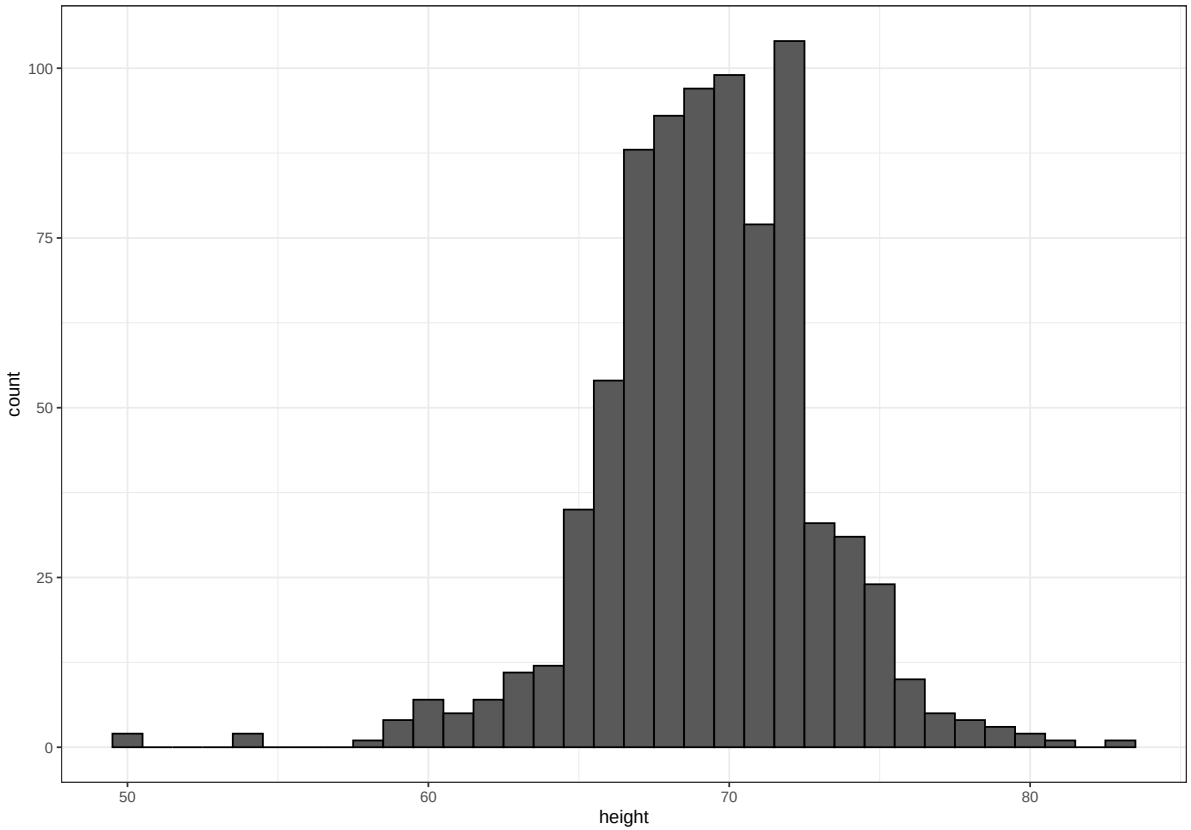
# Histograms

Histograms are often a better choice for these data. Histograms sacrifice just a bit of information to produce plots that are much easier to interpret.

A histogram divides the span of our data into non-overlapping bins of the same size. Then, for each bin, we count the number of values that fall in that interval. The histogram plots these counts as bars with the base of the bar defined by the intervals. Here is the histogram for the height data splitting the range of values into one inch intervals:
$(49.5, 50.5], (50.5, 51.5], (51.5, 52.5], (52.5, 53.5], \ldots, (82.5, 83.5]$

# Histograms

```
heights %>%
  filter(sex=="Male") %>%
  ggplot(aes(height)) +
  geom_histogram(binwidth = 1, color = "black")
```



# Histograms

A histogram is similar to a barplot, but it differs in that the x-axis is numerical, not categorical.

If we send this plot to ET, he will immediately learn some important properties about our data. First, the range of the data is from 50 to 84 with the majority (more than 95%) between 63 and 75 inches. Second, the heights are close to symmetric around 69 inches. Also, by adding up counts, ET could obtain a very good approximation of the proportion of the data in any interval.

Therefore, the histogram above is not only easy to interpret, but also provides almost all the information contained in the raw list of 812 heights with about 30 bin counts.

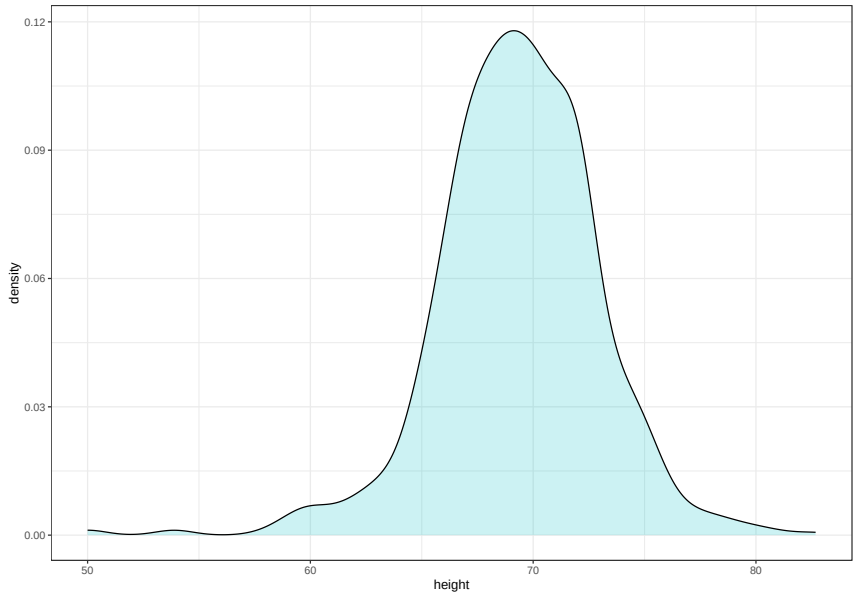
# Histograms

What information do we lose? Note that all values in each interval are treated the same when computing bin heights. So, for example, the histogram does not distinguish between 64, 64.1, and 64.2 inches. Given that these differences are almost unnoticeable to the eye, the practical implications are negligible and we were able to summarize the data to just 23 numbers.

# Smoothed Density

Smooth density plots are aesthetically more appealing than histograms:

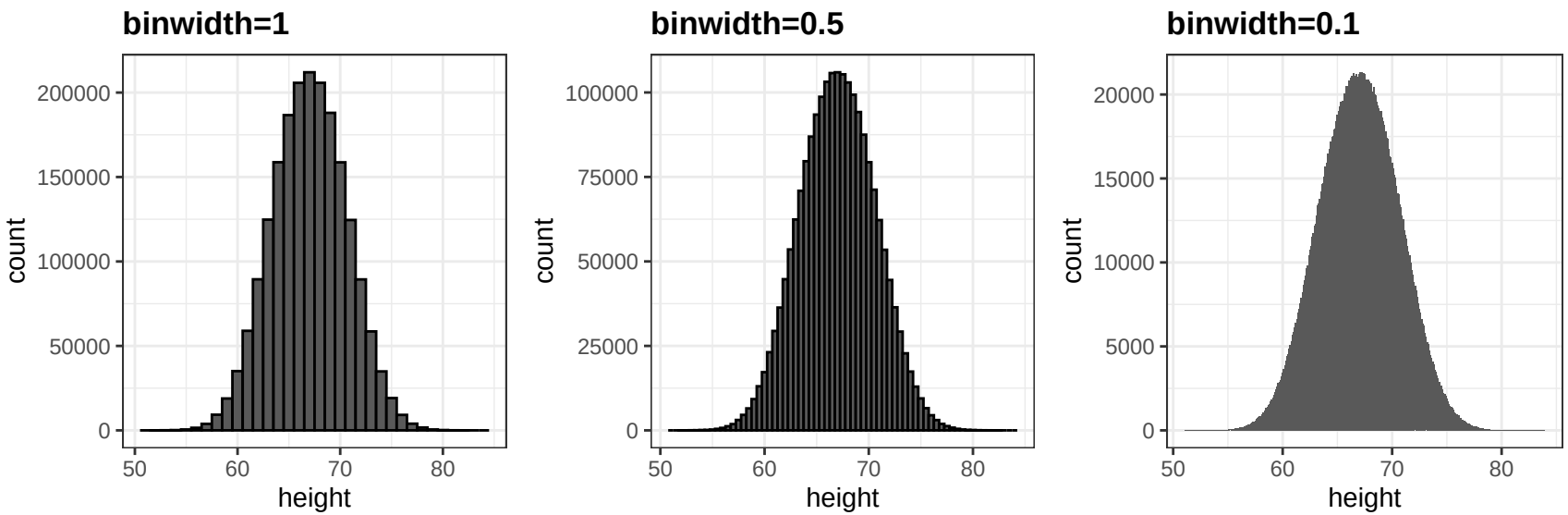
```
heights %>%
  filter(sex=="Male") %>%
  ggplot(aes(height)) +
  geom_density(alpha = .2,
               fill= "#00BFC4", color = 0)  +
  geom_line(stat='density')
```



# Smoothed Density

In this plot, we no longer have sharp edges at the interval boundaries and many of the local peaks have been removed. Also, the scale of the y-axis changed from counts to **density**.

A density is like a **smoothed** histogram if you had, say, 1,000,000 values, measured very precisely. The smaller we make the bins, the smoother the histogram gets:

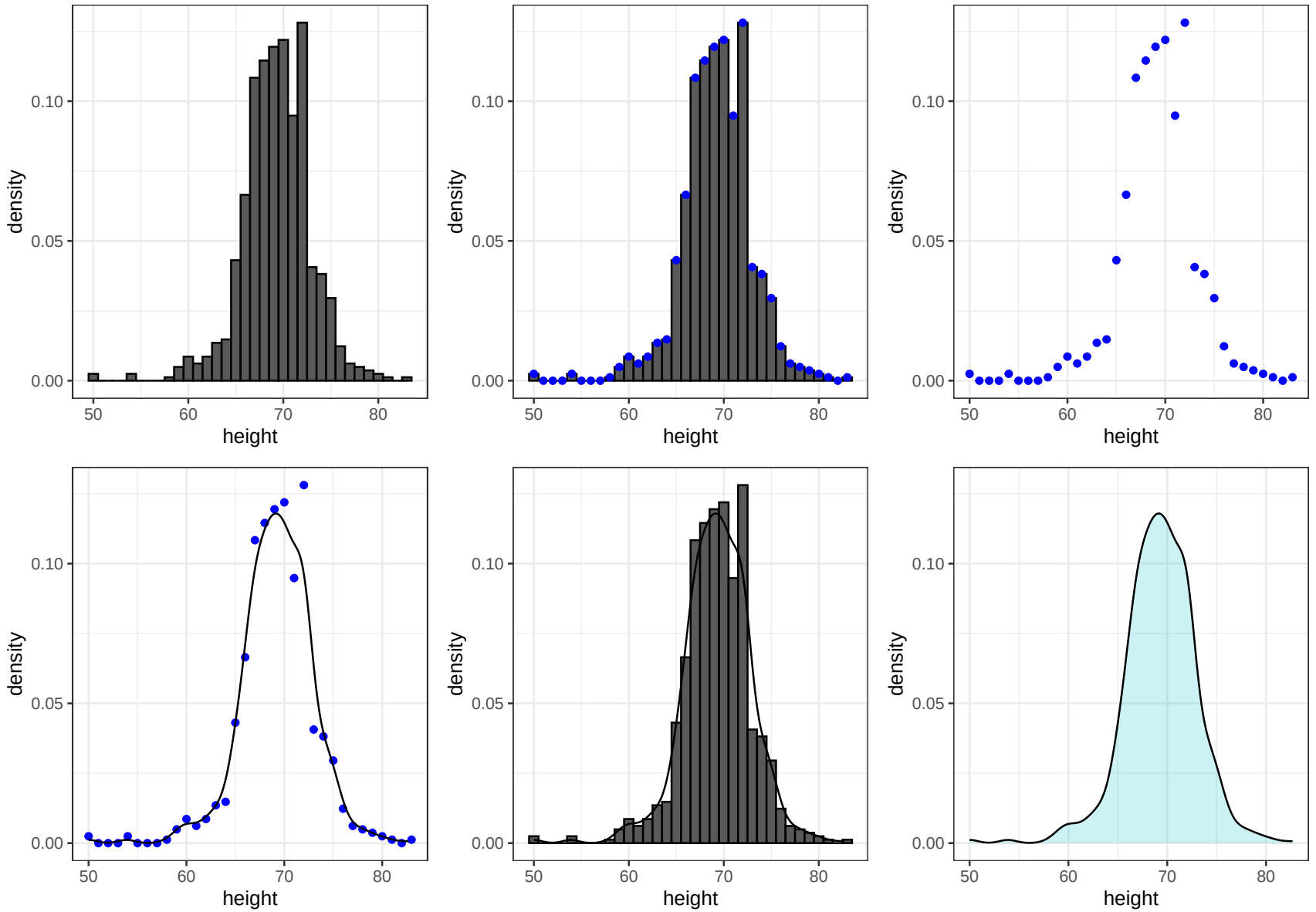


# Smoothed Density

Now, back to reality. We don't have millions of measurements. Instead, we have 812 and we can't make a histogram with very small bins.

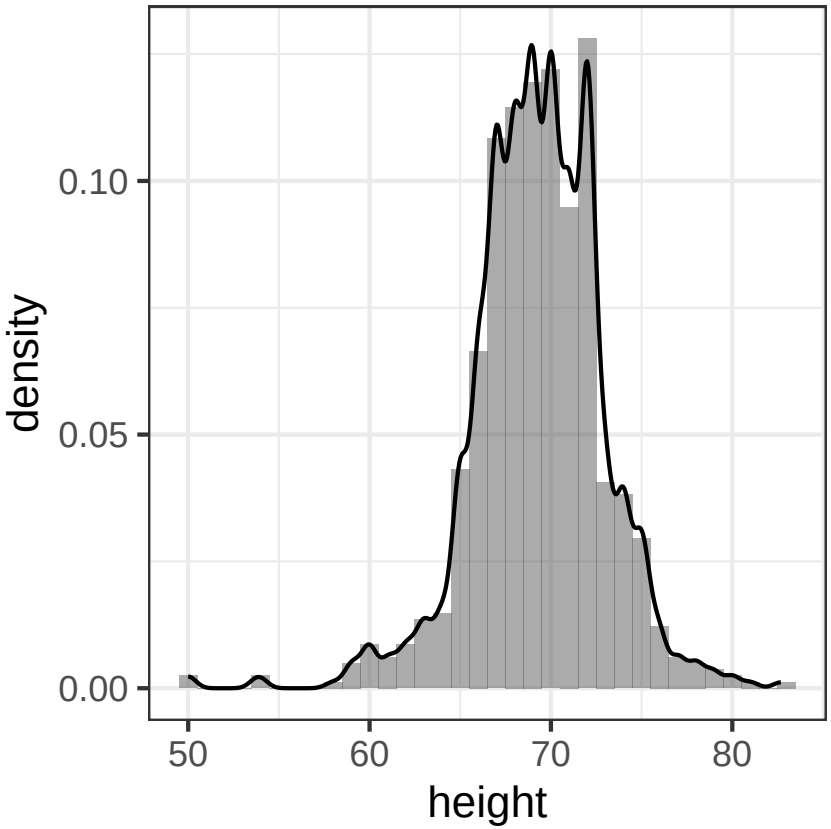
We therefore make a histogram, using bin sizes appropriate for our data and computing frequencies rather than counts, and we draw a smooth curve that goes through the tops of the histogram bars. The plots on the following slide demonstrate the steps that lead to a smooth density:

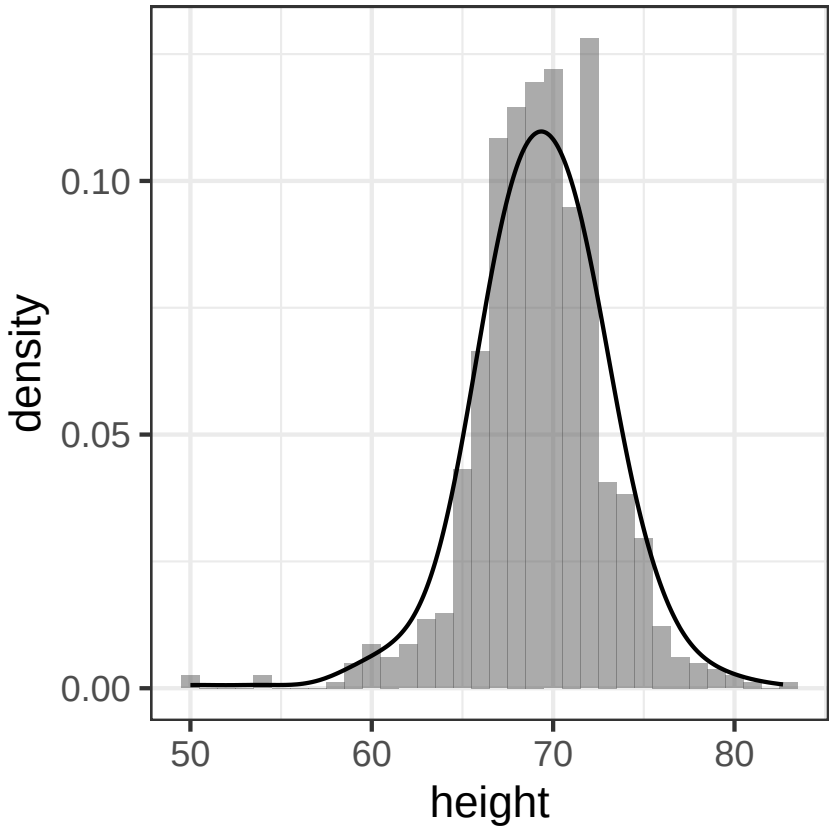
# Smoothed Density



# Smoothed Density

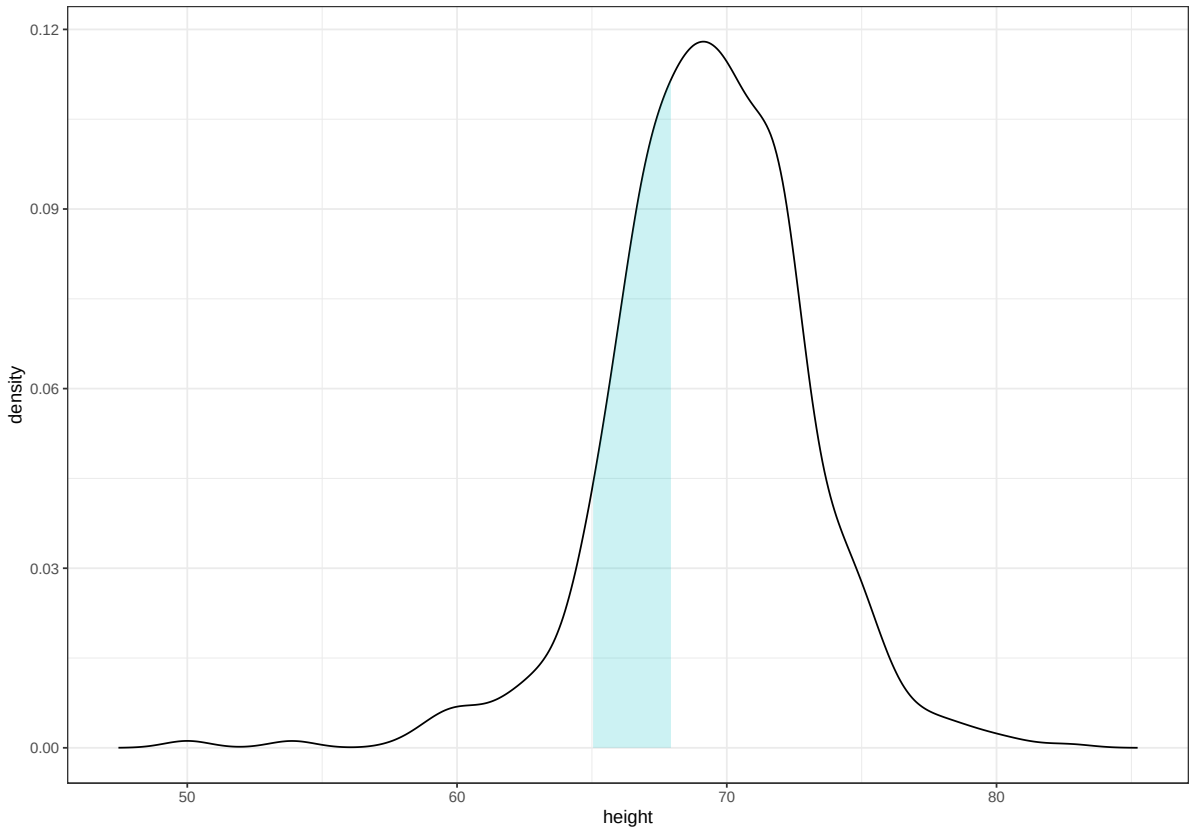
However, remember that **smooth** is a relative term. We can actually control the **smoothness** of the curve that defines the smooth density through an option in the function that computes the smooth density curve. For example:



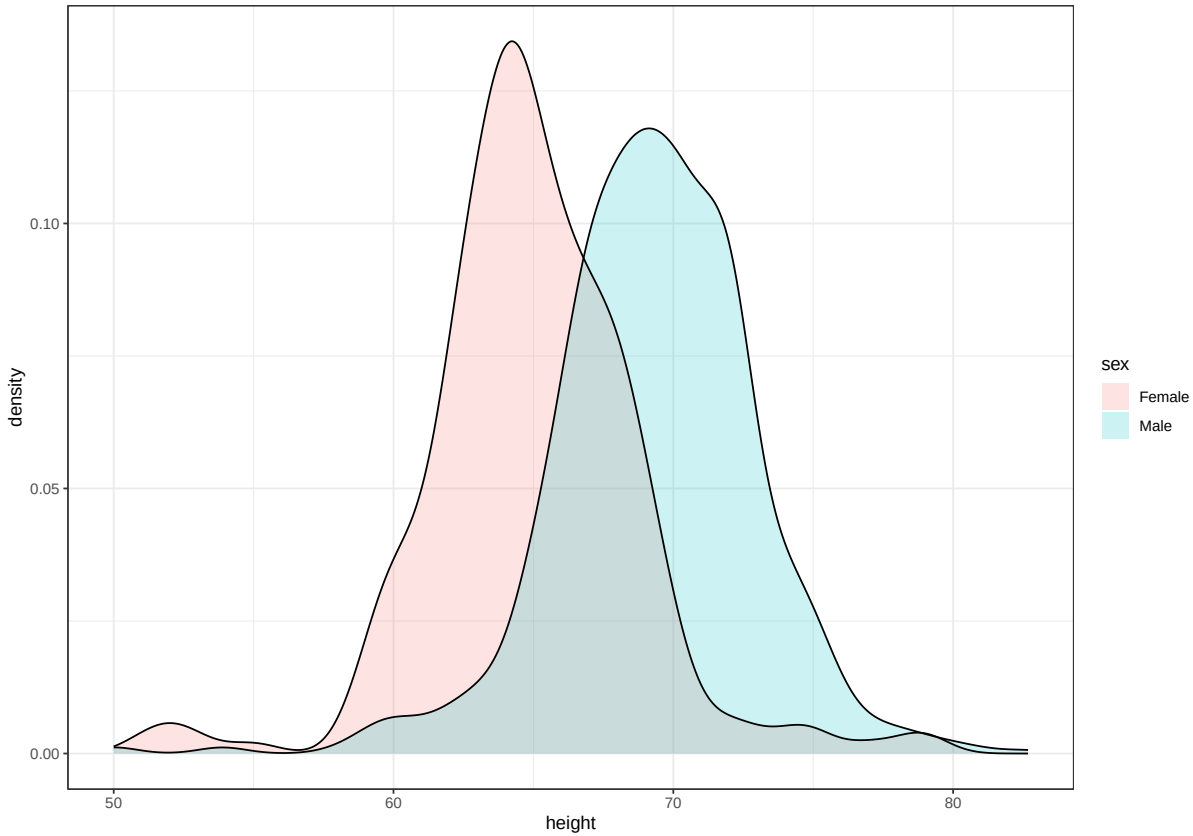
# Interpreting the y-axis

Note that interpreting the y-axis of a smooth density plot is not straightforward. It is scaled so that the area under the density curve adds up to 1. The best way to determine the proportion of data in that interval is by computing the proportion of the total area contained in that interval. For example, here are the proportion of values between 65 and 68:



# Densities Permit Stratification

As a final note, we point out that an advantage of smooth densities over histograms for visualization purposes is that densities make it easier to compare two distributions. This is in large part because the jagged edges of the histogram add clutter. Here is an example comparing male and female heights:



# Exercises

You are now ready for the Exercises 1-10 in the **ggplot2_additional_exercises.pdf** file!

# The normal distribution

The normal distribution, also known as the bell curve and as the Gaussian distribution, is one of the most famous mathematical concepts in history. A reason for this is that approximately normal distributions occur in many situations, including gambling winnings, heights, weights, blood pressure, standardized test scores, and experimental measurement errors. There are explanations for this, but we describe these later. Here we focus on how the normal distribution helps us summarize data.

# The Normal Distribution

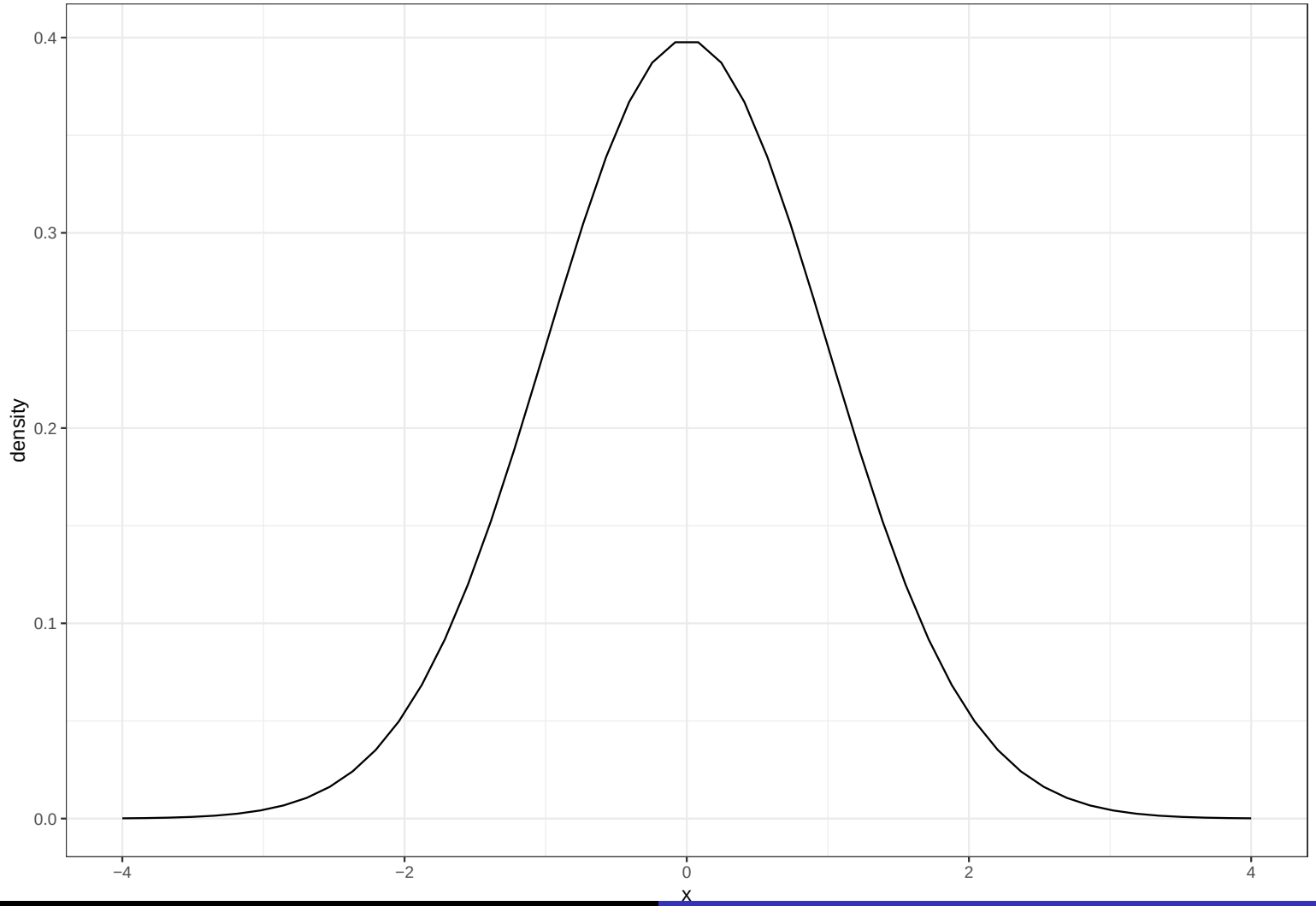
Rather than using data, the normal distribution is defined with a mathematical formula. For any interval $(a, b)$, the proportion of values in that interval can be computed using this formula:

$$\Pr(a < x < b) = \int_a^b \frac{1}{\sqrt{2\pi}s} e^{-\frac{1}{2}\left(\frac{x-m}{s}\right)^2} dx$$

You don't need to memorize or understand the details of the formula. But note that it is completely defined by just two parameters: $m$ and $s$. The rest of the symbols in the formula represent the interval ends that we determine, $a$ and $b$, and known mathematical constants $\pi$ and $e$. These two parameters, $m$ and $s$, are referred to as the *average* (also called the *mean*) and the *standard deviation* (SD) of the distribution, respectively.

# The Normal Distribution

The distribution is symmetric, centered at the average, and most values (about 95%) are within 2 SDs from the average. Here is a normal distribution with an average of 0 and SD 1:



# The Normal Distribution

Let's compute the values for the height for males which we will store in the object $x$:

```
index <- heights$sex == "Male"
x <- heights$height[index]
```

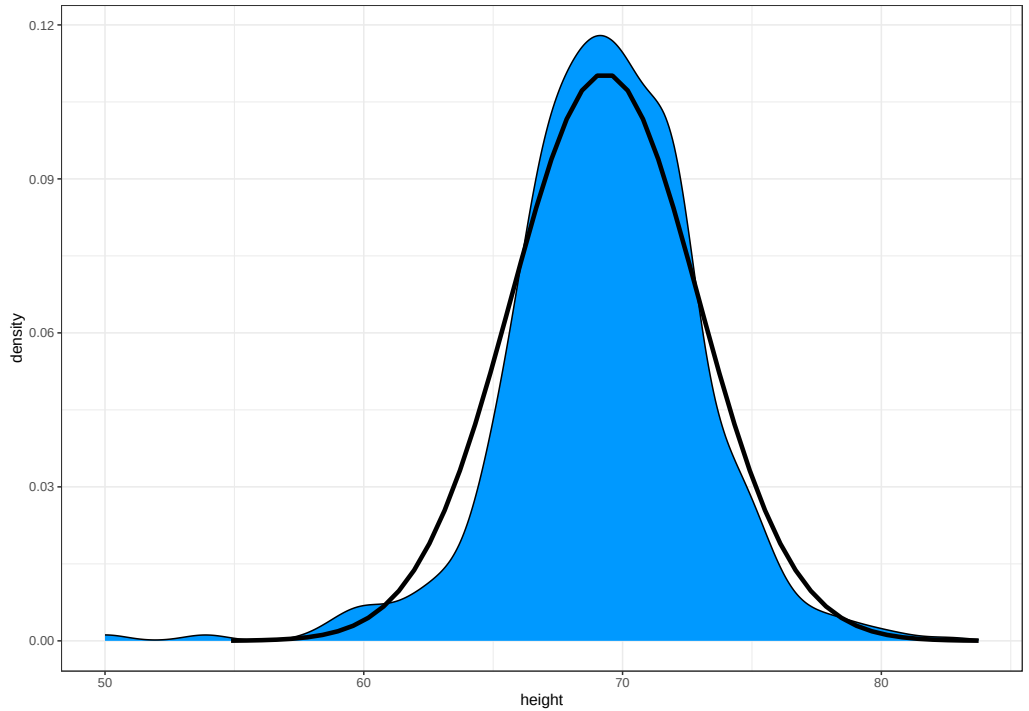
The pre-built functions `mean` and `sd` (note that for reasons explained in Section @ref(data-driven-model), `sd` divides by `length(x)-1` rather than `length(x)`) can be used here:

```
m <- mean(x)
s <- sd(x)
c(average = m, sd = s)
```

```
##   average        sd
## 69.314755  3.611024
```

# The Normal Distribution

Here is a plot of the smooth density and the normal distribution with mean $= 69.3$ and SD $= 3.6$ plotted as a black line with our student height smooth density in blue:




The normal distribution does appear to be quite a good approximation here.

# Quantile-quantile Plots

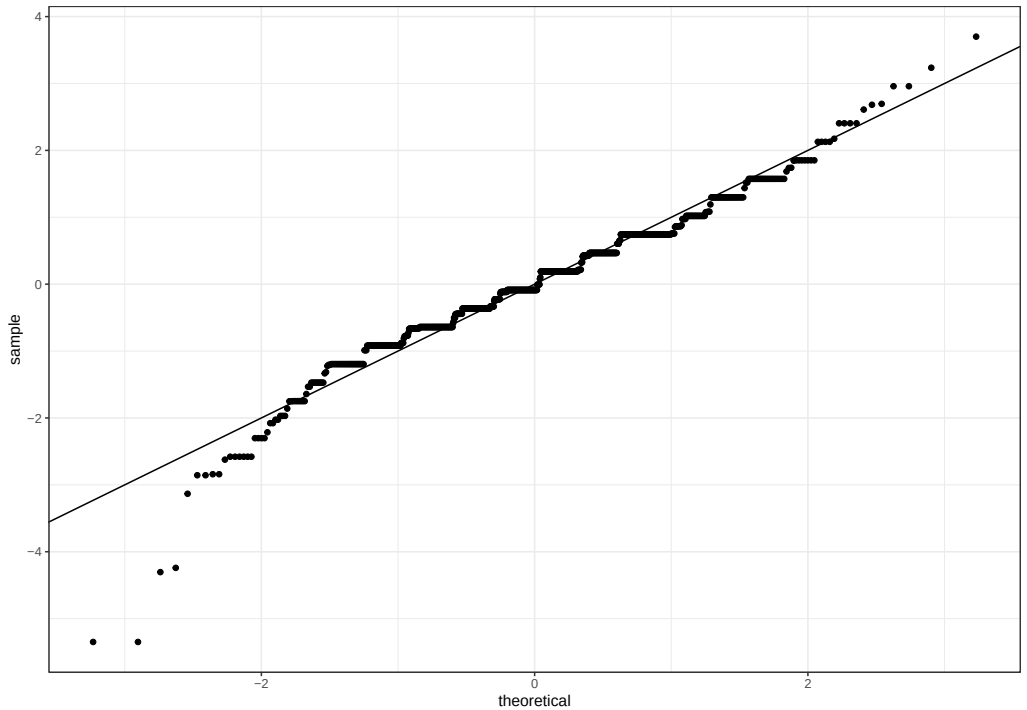
A systematic way to assess how well the normal distribution fits the data is to check if the observed and predicted proportions match. In general, this is the approach of the quantile-quantile plot (QQ-plot). The idea of a QQ-plot is that if your data is well approximated by normal distribution then the quantiles of your data should be similar to the quantiles of a normal distribution. To construct a QQ-plot, we do the following:

1. Define a vector of $m$ proportions $p_1, p_2, \ldots, p_m$.
2. Define a vector of quantiles $q_1, \ldots, q_m$ for your data for the proportions $p_1, \ldots, p_m$. We refer to these as the *sample quantiles*.
3. Define a vector of theoretical quantiles for the proportions $p_1, \ldots, p_m$ for a normal distribution with the same average and standard deviation as the data.
4. Plot the sample quantiles versus the theoretical quantiles.

# Quantile-quantile Plots

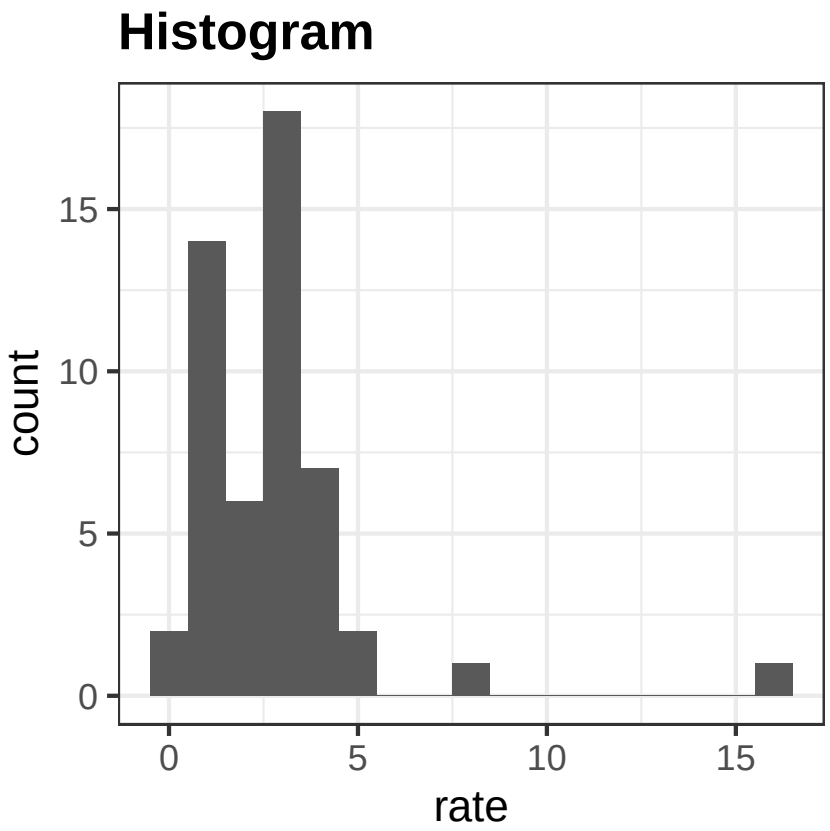
In practice we can use ggplot2 to generate a QQ plot:

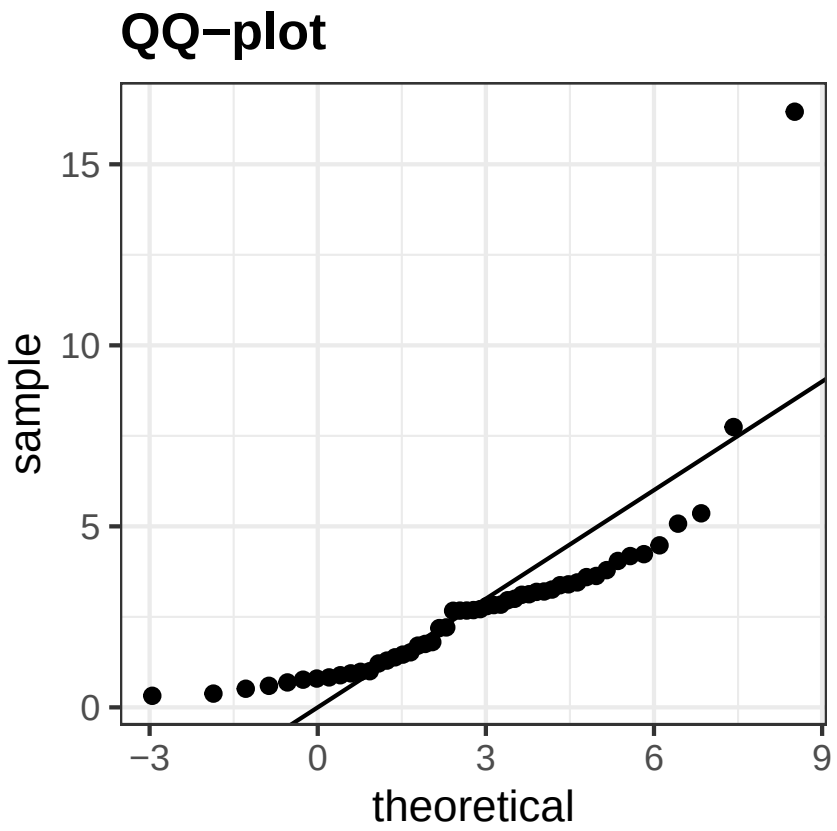
```
heights %>% filter(sex == "Male") %>%
  ggplot(aes(sample = scale(height))) +
  geom_qq() +
  geom_abline()
```



# Boxplots

To introduce boxplots we will go back to the US murder data. Suppose we want to summarize the murder rate distribution. Using the data visualization technique we have learned, we can quickly see that the normal approximation does not apply here:



# Boxplots

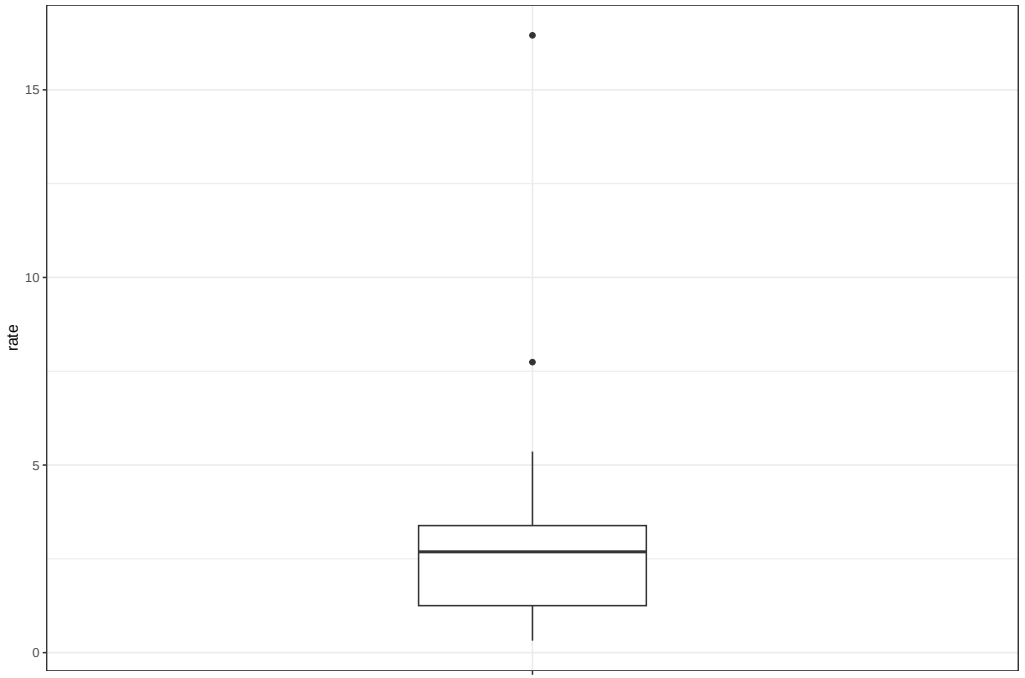
In this case, the histogram above or a smooth density plot would serve as a relatively succinct summary.

Now suppose those used to receiving just two numbers as summaries ask us for a more compact numerical summary.

Here Tukey offered some advice. Provide a five-number summary composed of the range along with the quartiles (the 25th, 50th, and 75th percentiles). Tukey further suggested that we ignore **outliers** when computing the range and instead plot these as independent points. We provide a detailed explanation of outliers later. Finally, he suggested we plot these numbers as a "box" with "whiskers".

# Boxplots

The box defined by the 25% and 75% percentile and the whiskers showing the range. The distance between these two is called the **interquartile range**. The two points are outliers according to Tukey's definition. The median is shown with a horizontal line. Today, we call these **boxplots**.

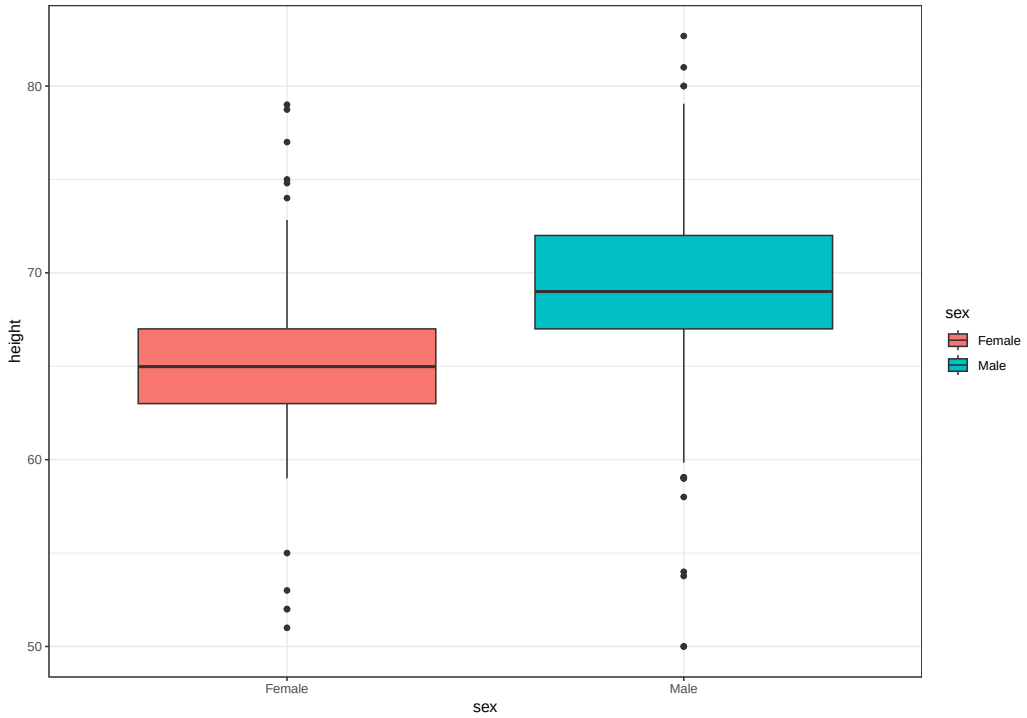


# Case study: describing student heights (continued)

Using the histogram, density plots, and QQ-plots, we have become convinced that the male height data is well approximated with a normal distribution. In this case, we report back to ET a very succinct summary: male heights follow a normal distribution with an average of 69.3 inches and a SD of 3.6 inches. With this information, ET will have a good idea of what to expect when he meets our male students. However, to provide a complete picture we need to also provide a summary of the female heights.

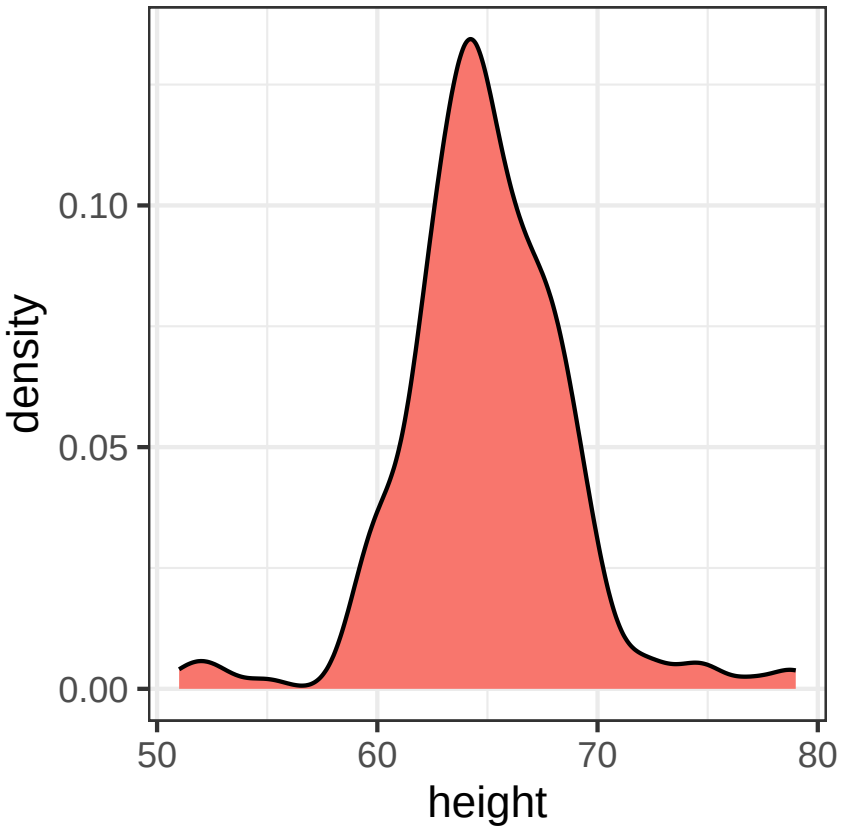
We learned that boxplots are useful when we want to quickly compare two or more distributions. Here are the heights for men and women:

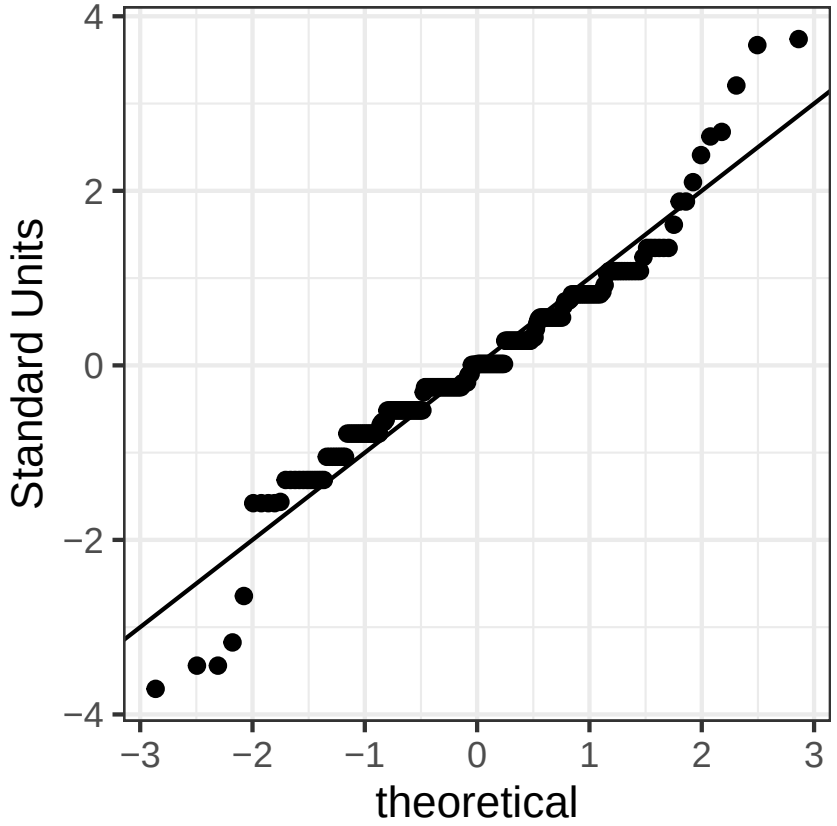
# Case study: describing student heights (continued)




The plot immediately reveals that males are, on average, taller than females. The standard deviations appear to be similar. But does the normal approximation also work for the female height data collected by the survey? We expect that they will follow a normal distribution, just like males. However, exploratory plots reveal that the approximation is not as useful:

# Case study: describing student heights (continued)



# Case study: describing student heights (continued)

Regarding the five smallest values, note that these values are:

```
heights %>% filter(sex == "Female") %>%
  top_n(5, desc(height)) %>%
  pull(height)
```

```
## [1] 51 53 55 52 52
```

Because these are reported heights, a possibility is that the student meant to enter `5'1"`, `5'2"`, `5'3"` or `5'5"`.

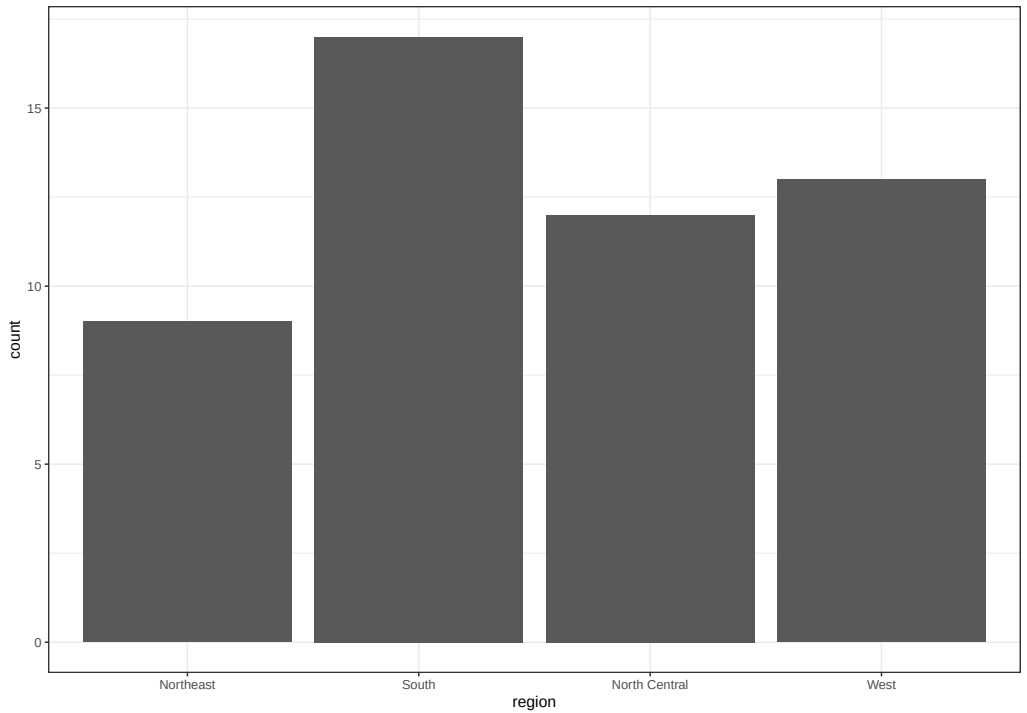
# Exercises

You are now ready for the Exercises 11-25 in the **ggplot2_additional_exercises.pdf** file!

# Barplots

To generate a barplot we can use the `geom_bar` geometry. The default is to count the number of each category and draw a bar. Here is the plot for the regions of the US.

```
murders %>% ggplot(aes(region)) + geom_bar()
```



# Barplots

We often already have a table with a distribution that we want to present as a barplot. Here is an example of such a table:

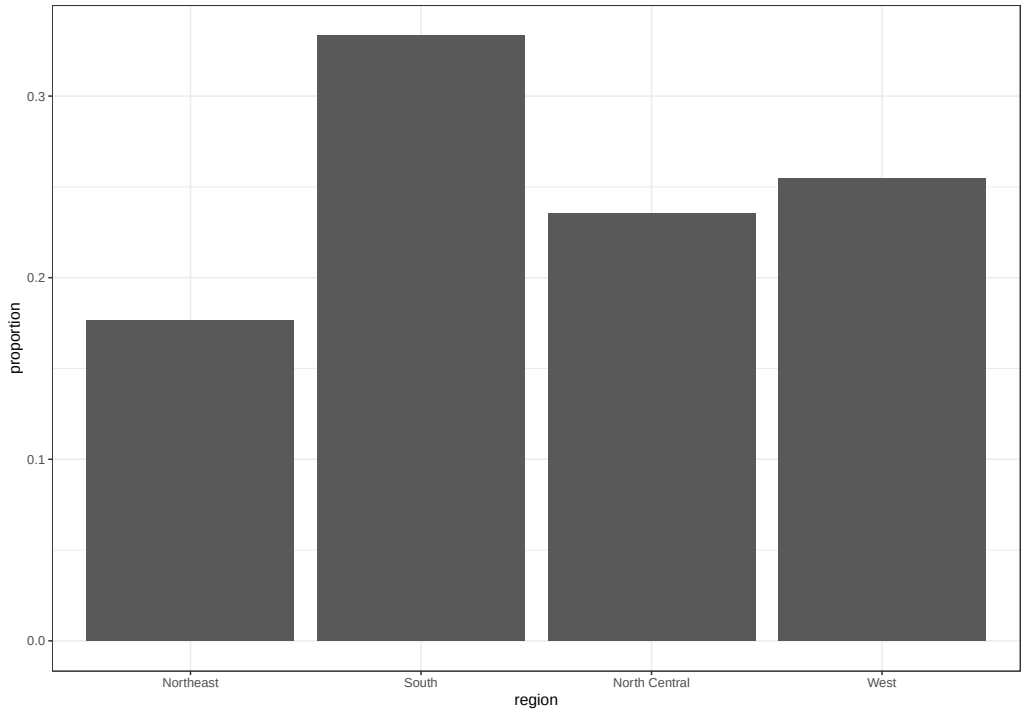
```
data(murders)
tab <- murders %>%
  count(region) %>%
  mutate(proportion = n/sum(n))
tab
```

```
##          region  n proportion
## 1     Northeast  9  0.1764706
## 2         South 17  0.3333333
## 3 North Central 12  0.2352941
## 4          West 13  0.2549020
```

# Barplots

We no longer want `geom_bar` to count, but rather just plot a bar to the height provided by the `proportion` variable. For this we need to provide `x` (the categories) and `y` (the values) and use the `stat="identity"` option.

```
tab %>% ggplot(aes(region, proportion)) +
  geom_bar(stat = "identity")
```



# Histograms

To generate histograms we use `geom_histogram`. By looking at the help file for this function, we learn that the only required argument is `x`, the variable for which we will construct a histogram. We dropped the `x` because we know it is the first argument. The code looks like this:

```
heights %>%
  filter(sex == "Female") %>%
  ggplot(aes(height)) +
  geom_histogram()
```

# Histograms

If we run the code above, it gives us a message:

> *`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.*

# Histograms

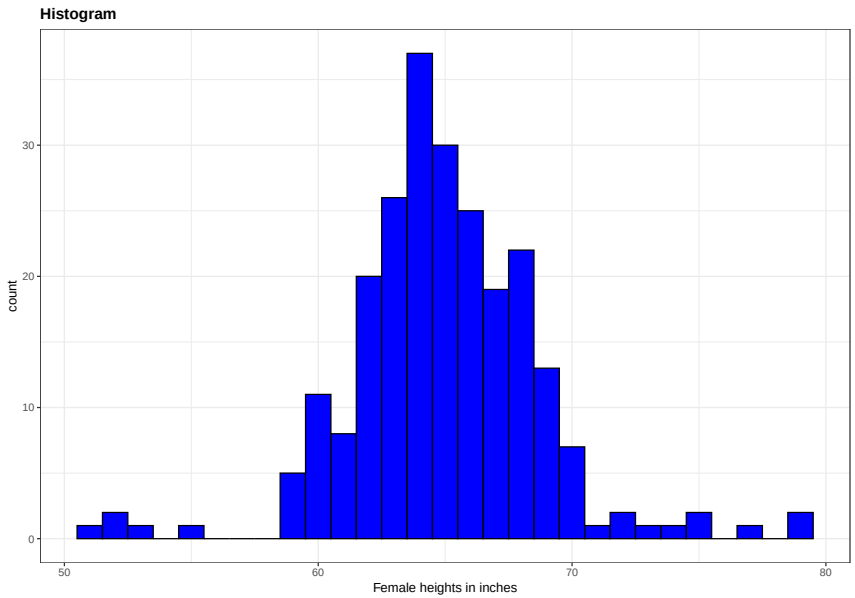
We previously used a bin size of 1 inch, so the code looks like this:

```
heights %>%
  filter(sex == "Female") %>%
  ggplot(aes(height)) +
  geom_histogram(binwidth = 1)
```

# Histograms

Finally, if for aesthetic reasons we want to add color, we use the arguments described in the help file. We also add labels and a title:

```
heights %>%
  filter(sex == "Female") %>%
  ggplot(aes(height)) +
  geom_histogram(binwidth = 1, fill = "blue", col = "black") +
  xlab("Female heights in inches") +
  ggtitle("Histogram")
```



# Density plots

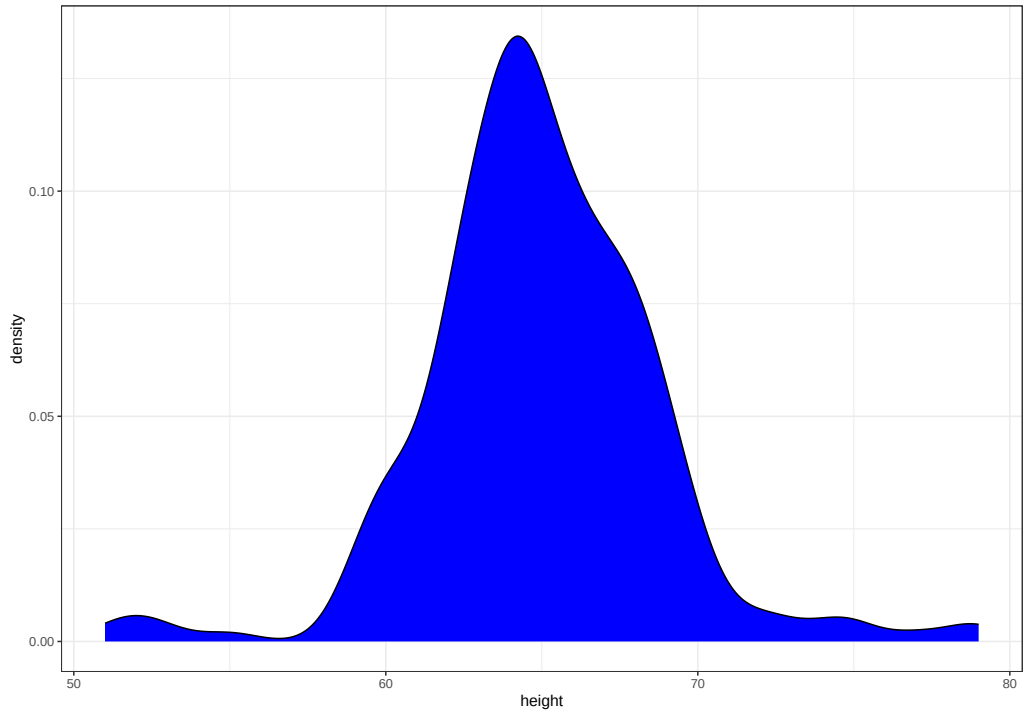
To create a smooth density, we use the `geom_density`. To make a smooth density plot with the data previously shown as a histogram we can use this code:

```
heights %>%
  filter(sex == "Female") %>%
  ggplot(aes(height)) +
  geom_density()
```

# Density plots

To fill in with color, we can use the `fill` argument.

```
heights %>%
  filter(sex == "Female") %>%
  ggplot(aes(height)) +
  geom_density(fill="blue")
```



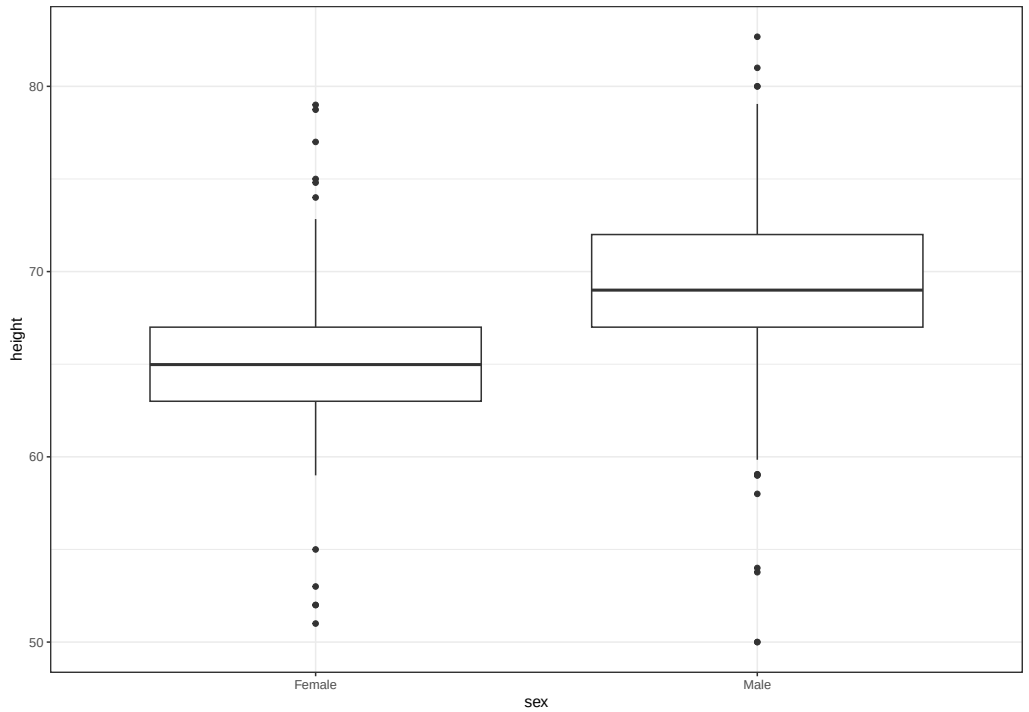
# Density plots

To change the smoothness of the density, we use the `adjust` argument to multiply the default value by that `adjust`. For example, if we want the bandwidth to be twice as big we use:

```
heights %>%
  filter(sex == "Female") +
  geom_density(fill="blue", adjust = 2)
```

# Boxplots

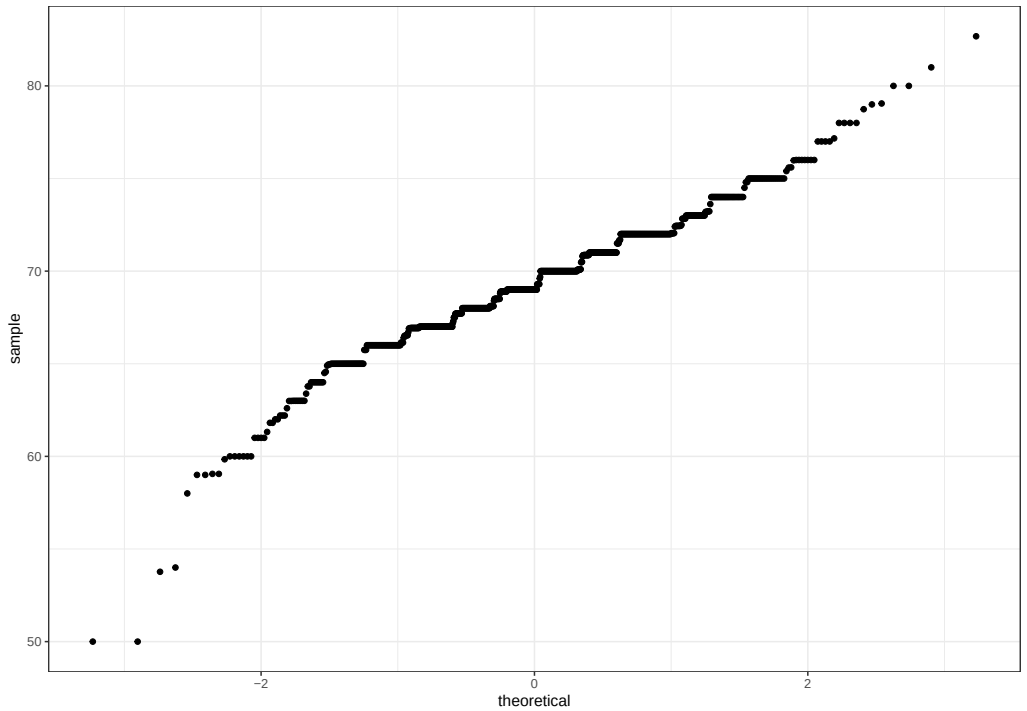
The geometry for boxplot is `geom_boxplot`. As discussed, boxplots are useful for comparing distributions. For example, below are the previously shown heights for women, but compared to men. For this geometry, we need arguments `x` as the categories, and `y` as the values.



# QQ-plots

For qq-plots we use the geom_qq geometry. From the help file, we learn that we need to specify the sample (we will learn about samples in a later chapter). Here is the qqplot for men heights.

```
heights %>% filter(sex=="Male") %>%
  ggplot(aes(sample = height)) +
  geom_qq()
```



# Images

Images were not needed for the concepts described in this chapter, but we will use images in Section @ref(vaccines), so we introduce the two geometries used to create images: **geom_tile** and **geom_raster**. They behave similarly; to see how they differ, please consult the help file. To create an image in ggplot1 we need a data frame with the x and y coordinates as well as the values associated with each of these. Here is a data frame.

```
x <- expand.grid(x = 1:12, y = 1:10) %>%
  mutate(z = 1:120)
```

# Images

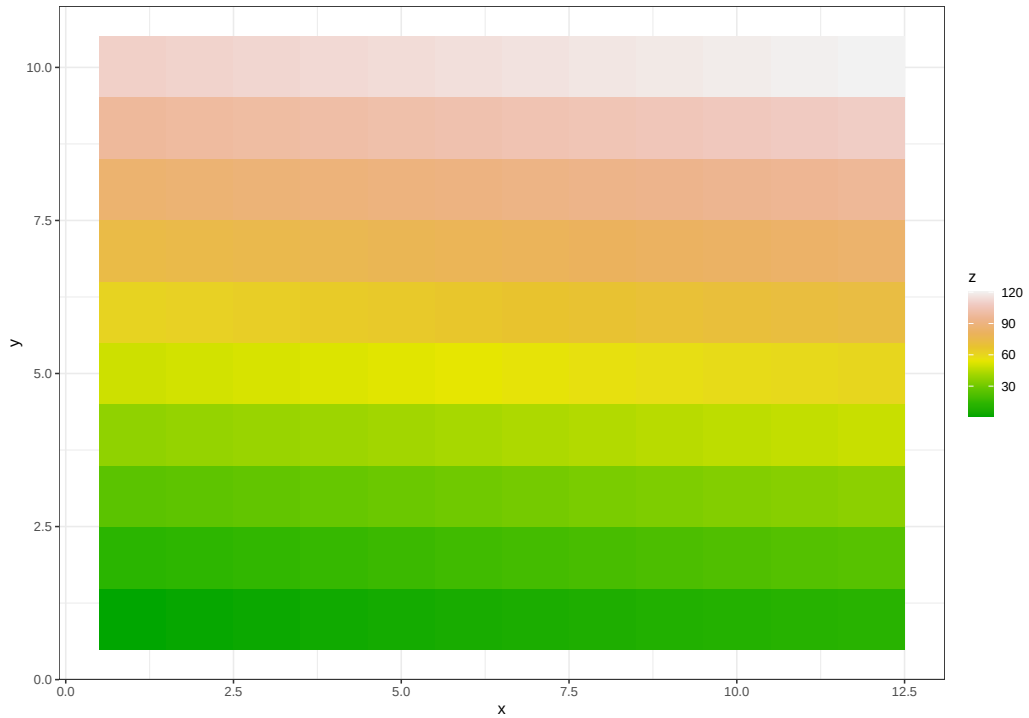
Note that this is the tidy version of a matrix, `matrix(1:120, 12, 10)`. To plot the image we use the following code:

```
x %>% ggplot(aes(x, y, fill = z)) +
  geom_raster()
```

# Images

With these images you will often want to change the color scale. This can be done through the `scale_fill_gradientn` layer.

```
x %>% ggplot(aes(x, y, fill = z)) +
  geom_raster() +
  scale_fill_gradientn(colors =  terrain.colors(10))
```



# Quick plots

In Section @ref(qplot) we introduced `qplot` as a useful function when we need to make a quick scatterplot. We can also use `qplot` to make histograms, density plots, boxplot, qqplots and more. Although it does not provide the level of control of `ggplot`, `qplot` is definitely useful as it permits us to make a plot with a short snippet of code.

# Quick plots

Suppose we have the female heights in an object `x`:

```
x <- heights %>%
  filter(sex=="Male") %>%
  pull(height)
```

# Quick plots

To make a quick histogram we can use:

```
qplot(x)
```

# Quick plots

The function guesses that we want to make a histogram because we only supplied one variable. In Section @ref(qplot) we saw that if we supply `qplot` two variables, it automatically makes a scatterplot.

To make a quick qqplot you have to use the `sample` argument. Note that we can add layers just as we do with `ggplot`.

```
qplot(sample = scale(x)) + geom_abline()
```

# Quick plots

If we supply a factor and a numeric vector, we obtain a plot like the one below. Note that in the code below we are using the `data` argument. Because the data frame is not the first argument in `qplot`, we have to use the dot operator.

```
heights %>% qplot(sex, height, data = .)
```

# Quick plots

We can also select a specific geometry by using the geom argument. So to convert the plot above to a boxplot, we use the following code:

```
heights %>% qplot(sex, height, data = ., geom = "boxplot")
```

# Quick plots

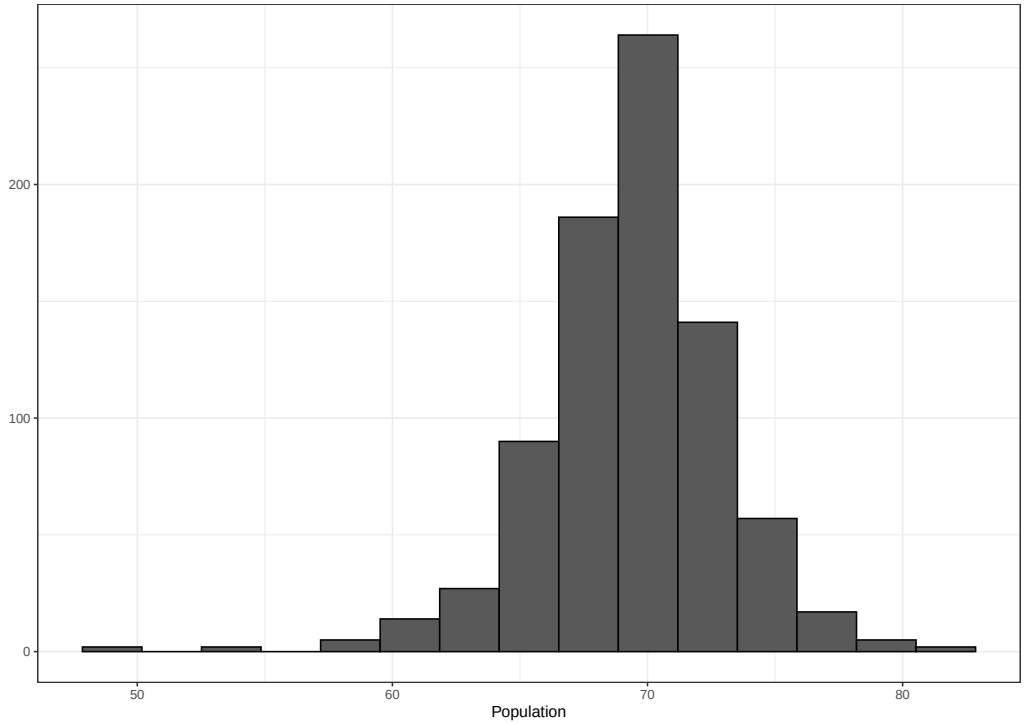
We can also use the `geom` argument to generate a density plot instead of a histogram:

```
qplot(x, geom = "density")
```

# Quick plots

Although not as much as with ggplot, we do have some flexibility to improve the results of qplot. Looking at the help file we see several ways in which we can improve the look of the histogram above. Here is an example:

```
qplot(x, bins=15, color = I("black"), xlab = "Population")
```



# Quick plots

**Technical note**: The reason we use `I("black")` is because we want `qplot` to treat `"black"` as a character rather than convert it to a factor, which is the default behavior within `aes`, which is internally called here. In general, the function `I` is used in R to say "keep it as it is".

# Exercises

# Session Info

```
sessionInfo()
```

```
## R version 4.3.2 (2023-10-31)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.5.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LAPACK vers
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/Denver
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] gridExtra_2.3   dslabs_0.7.6    lubridate_1.9.3 forcats_1.0.0
##  [5] stringr_1.5.1   dplyr_1.1.3     purrr_1.0.2     readr_2.1.4
##  [9] tidyr_1.3.0     tibble_3.2.1    ggplot2_3.4.4   tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] gtable_0.3.4      crayon_1.5.2      compiler_4.3.2    tidyselect_1.2.0
##  [5] scales_1.2.1      yaml_2.3.7        fastmap_1.1.1     R6_2.5.1
##  [9] labeling_0.4.3    generics_0.1.3    knitr_1.45        munsell_0.5.0
## [13] pillar_1.9.0      tzdb_0.4.0        rlang_1.1.2       utf8_1.2.4
## [17] stringi_1.8.1     xfun_0.41         timechange_0.2.0  cli_3.6.1
## [21] withr_2.5.2       magrittr_2.0.3    digest_0.6.33     grid_4.3.2
## [25] rstudioapi_0.15.0 hms_1.1.3         lifecycle_1.0.4   vctrs_0.6.4
## [29] evaluate_0.23     glue_1.6.2        farver_2.1.1      fansi_1.0.5
## [33] colorspace_2.1-0  rmarkdown_2.25    tools_4.3.2       pkgconfig_2.0.3
## [37] htmltools_0.5.7
```