We've used ggplots throughout this blog series, but today, I want to introduce another package that helps you customize scales on your ggplots – the scales package. I use this package most frequently to format scales as percent. There aren't a lot of good ways to use percents with my dataset, but one example would be to calculate the percentage each book contributes to the total pages I read in 2019.

```
library(tidyverse)
```

```
## -- Attaching packages ------------------------------------------- tidyverse 1.3.0 --
```

```
##  ggplot2 3.2.1      purrr   0.3.3
##  tibble  2.1.3      dplyr   0.8.3
##  tidyr   1.0.0      stringr 1.4.0
##  readr   1.3.1      forcats 0.4.0
```

```
## -- Conflicts ---------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
reads2019 <- read_csv("~/Downloads/Blogging A to Z/SaraReads2019_allrated.csv",
                      col_names = TRUE)
```

```
## Parsed with column specification:
## cols(
##   Title = col_character(),
##   Pages = col_double(),
##   date_started = col_character(),
##   date_read = col_character(),
##   Book.ID = col_double(),
##   Author = col_character(),
##   AdditionalAuthors = col_character(),
##   AverageRating = col_double(),
##   OriginalPublicationYear = col_double(),
##   read_time = col_double(),
##   MyRating = col_double(),
##   Gender = col_double(),
##   Fiction = col_double(),
##   Childrens = col_double(),
##   Fantasy = col_double(),
##   SciFi = col_double(),
##   Mystery = col_double(),
##   SelfHelp = col_double()
## )
```

```
reads2019 <- reads2019 %>%
  mutate(perpage = Pages/sum(Pages))
```

The new variable, perpage, is a proportion. But if I display those data with a figure, I want them to be percentages instead. Here's how to do that. (If you don't already have the scales package, add install.packages("scales") at the beginning of this code.)

```
library(scales)
```

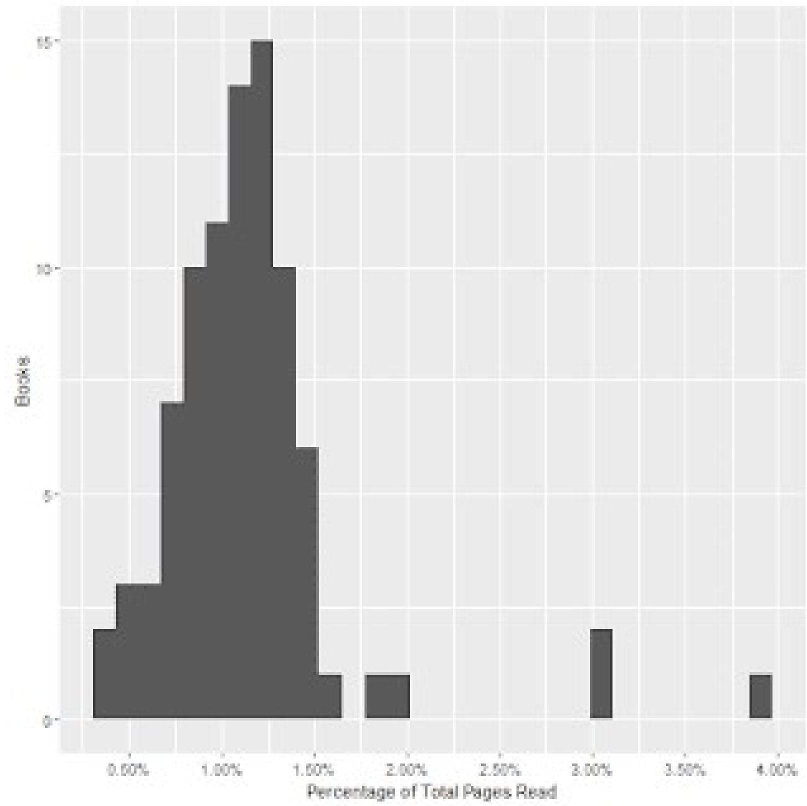
```
##
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:purrr':
##
```

```
##     discard

## The following object is masked from 'package:readr':
##
##     col_factor
```

```
reads2019 %>%
  ggplot(aes(perpage)) +
  geom_histogram() +
  scale_x_continuous(labels = percent, breaks = seq(0,.05,.005)) +
  xlab("Percentage of Total Pages Read") +
  ylab("Books")
```

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



You need to make sure you load the scales package before you add the labels = percent attribute, or you'll get an error message. Alternatively, you can tell R to use the scales package just for this attribute by adding scales:: before percent. This trick becomes useful when you have lots of packages loaded that use the same function names, because R will use the most recently loaded package for that function, and mask it from any other packages.

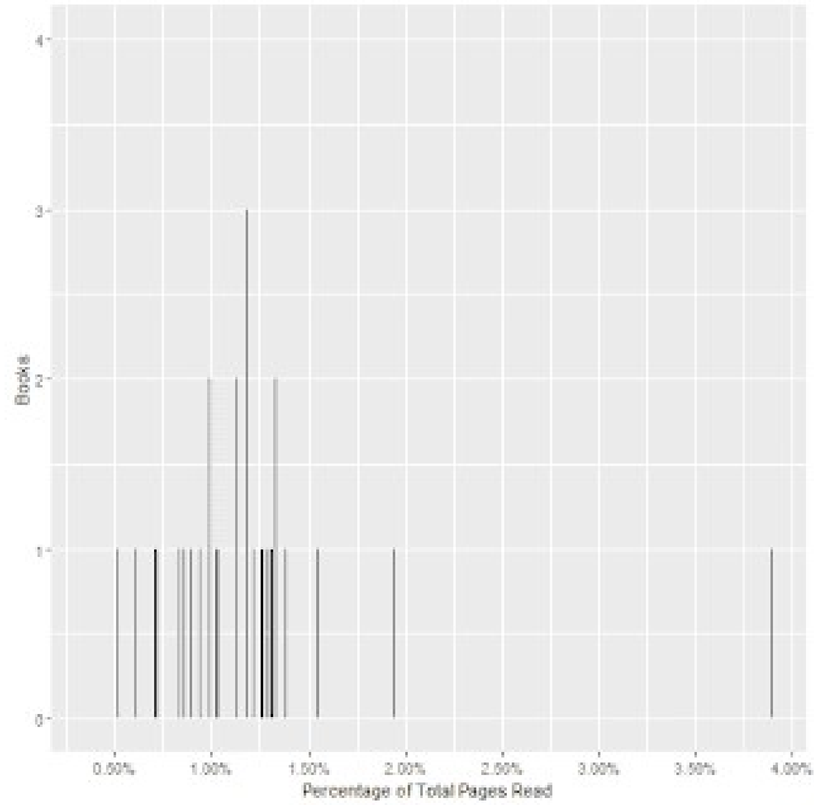
This post also seems like a great opportunity to hop on my statistical highhorse and talk about the difference between a histogram and a bar chart. Why is this important? With everything going on in the world – pandemics, political elections, etc. – I've seen lots of comments on others' intelligence, many of which show a misunderstanding of the most well-known histogram: the standard normal curve. You see, raw data, even from a huge number of people and *even* on a standardized test, like a cognitive ability (aka: IQ) test, is **never** as clean or pretty as it appears in a histogram.

Histograms use a process called "binning", where ranges of scores are combined to form one of the bars. The bins can be made bigger (including a larger range of scores) or smaller, and smaller bins will start showing the jagged nature of most data, even so-called normally distributed data.

As one example, let's show what my percent figure would look like as a bar chart instead of a histogram (like the one above).

```
reads2019 %>%
```

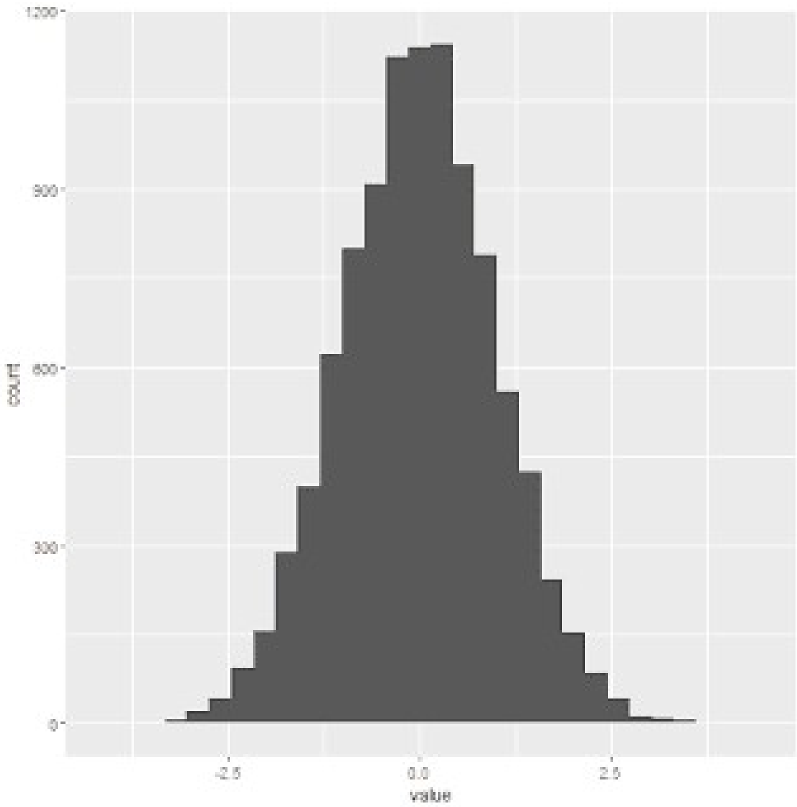
```
  ggplot(aes(perpage)) +
  geom_bar() +
  scale_x_continuous(labels = percent, breaks = seq(0,.05,.005)) +
  xlab("Percentage of Total Pages Read") +
  ylab("Books")
```



As you can see, lots of books were binned together for the histogram. I can customize the number of bins in my histogram, but unless I set it to give one bin to each x value, the result will be much cleaner than the bar chart. The same is true for cognitive ability scores. Each bar is a bin, and that bin contains a range of values. So when we talk about scores on a standardized test, we're really referring to a range of scores.

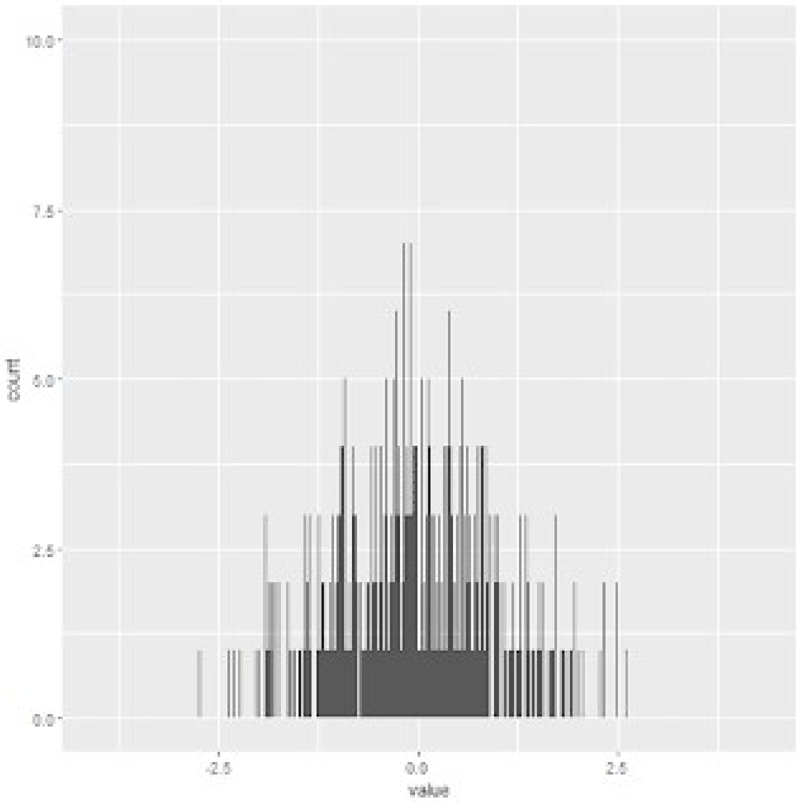
Now, my reading dataset is small – only 87 observations. What happens if I generate a large, random dataset?

```
set.seed(42)

test <- tibble(ID = c(1:10000),
               value = rnorm(10000))

test %>%
  ggplot(aes(value)) +
  geom_histogram()

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



See that “stat_bin()” warning message? It’s telling me that there are 30 bins, so R divided up the range of scores into 30 equally sized bins. What happens when I increase the number of bins? Let’s go really crazy and have it create one bin for each score value.

```
library(magrittr)

##
## Attaching package: 'magrittr'

## The following object is masked from 'package:purrr':
##
##     set_names

## The following object is masked from 'package:tidyr':
##
##     extract

test %$% n_distinct(value)

## [1] 10000

test %>%
  ggplot(aes(value)) +
  geom_histogram(bins = 10000)
```



Not nearly so pretty, is it? Mind you, this is 10,000 values randomly generated to follow the normal distribution. When you give each value a bin, it doesn't look very normally distributed.

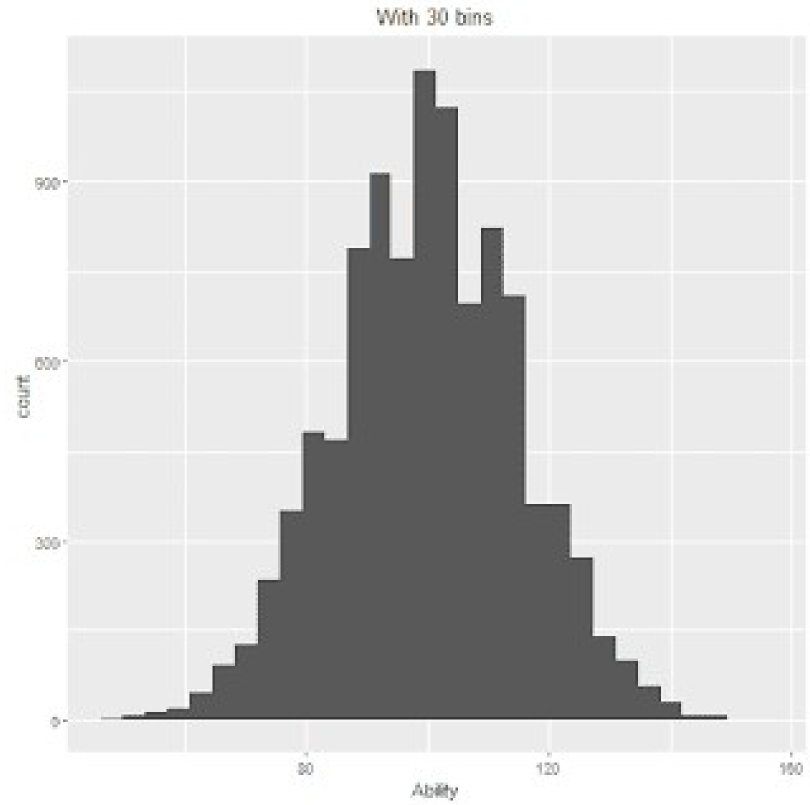
How about if we mimic cognitive ability scores, with a mean of 100 and a standard deviation of 15? I'll even force it to have whole numbers, so we don't have decimal places to deal with.

```
CogAbil <- tibble(Person = c(1:10000),
                  Ability = rnorm(10000, mean = 100, sd = 15))

CogAbil <- CogAbil %>%
  mutate(Ability = round(Ability, digits = 0))

CogAbil %$%
  n_distinct(Ability)
```

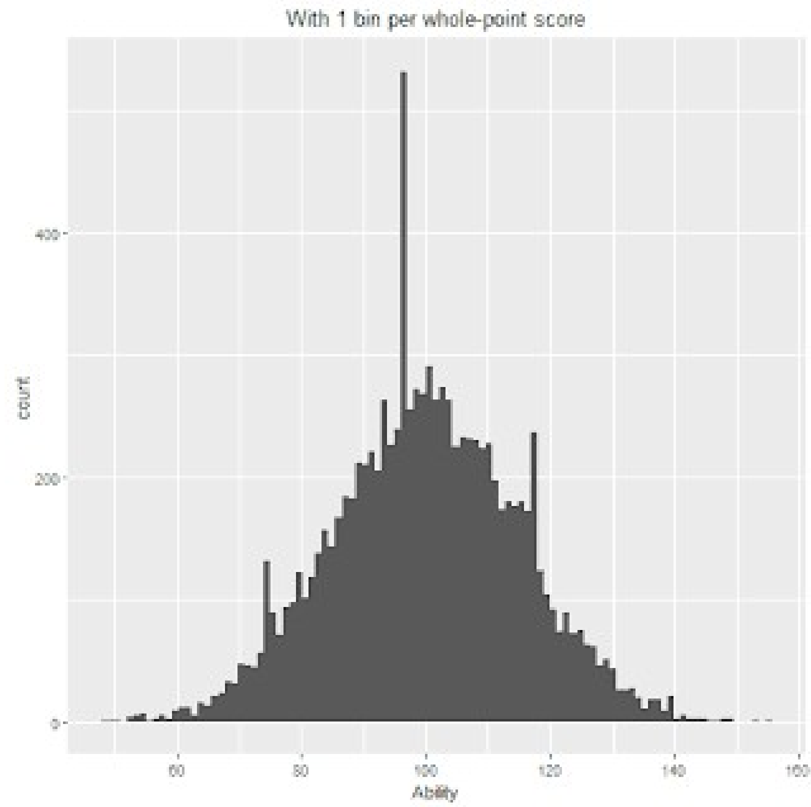
```
## [1] 103
```

```
CogAbil %>%
  ggplot(aes(Ability)) +
  geom_histogram() +
  labs(title = "With 30 bins") +
  theme(plot.title = element_text(hjust = 0.5))
```

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




```
CogAbil %>%
  ggplot(aes(Ability)) +
  geom_histogram(bins = 103) +
  labs(title = "With 1 bin per whole-point score") +
  theme(plot.title = element_text(hjust = 0.5))
```



(Now, there's more that goes into developing a cognitive ability test, because the original scale of the test (raw scores) differ from the standardized scale that is applied to turn raw scores into one with a mean of 100 and standard deviation of 15. That's where an entire field's worth of knowledge (psychometrics) comes in.)