

IS5 in R: The Standard Deviation as a Ruler and the Normal Model (Chapter 5)

Margaret Chien and Nicholas Horton (nhorton@amherst.edu)

July 19, 2018

Introduction and background

This document is intended to help describe how to undertake analyses introduced as examples in the Fifth Edition of *Intro Stats* (2018) by De Veaux, Velleman, and Bock. More information about the book can be found at http://wps.aw.com/aw_deveaux_stats_series. This file as well as the associated R Markdown reproducible analysis source file used to create it can be found at <http://nhorton.people.amherst.edu/is5>.

This work leverages initiatives undertaken by Project MOSAIC (<http://www.mosaic-web.org>), an NSF-funded effort to improve the teaching of statistics, calculus, science and computing in the undergraduate curriculum. In particular, we utilize the `mosaic` package, which was written to simplify the use of R for introductory statistics courses. A short summary of the R needed to teach introductory statistics can be found in the `mosaic` package vignettes (<http://cran.r-project.org/web/packages/mosaic>). A paper describing the `mosaic` approach was published in the *R Journal*: <https://journal.r-project.org/archive/2017/RJ-2017-024>.

Chapter 5: The Standard Deviation as a Ruler and the Normal Model

```
library(mosaic)
library(readr)
library(janitor)
WomenHeptathlon2016 <-
  read_csv("http://nhorton.people.amherst.edu/is5/data/Womens_Heptathlon_2016.csv") %>%
  clean_names()
```

```
## Parsed with column specification:
## cols(
##   `First Name` = col_character(),
##   `Last Name` = col_character(),
##   `200m` = col_double(),
##   LongJump = col_double(),
##   `800m` = col_double(),
##   HighJump = col_double(),
##   `100m.hurdles` = col_double(),
##   Javelin = col_double(),
##   ShotPut = col_double()
## )
```

By default, `read_csv()` prints the variable names. These messages can be suppressed using the `message = FALSE` code chunk option to save space and improve readability.

Here we use the `clean_names()` function from the `janitor` package to sanitize the names of the columns (which would otherwise contain special characters or whitespace).

```
# page 123
favstats(~ long_jump, data = WomenHeptathlon2016)
```

```
##   min    Q1 median    Q3   max    mean      sd  n missing
```

```
## 5.51 6.08 6.19 6.31 6.58 6.169655 0.2474655 29 2
favstats(~ x200m, data = WomenHeptathlon2016)

## min Q1 median Q3 max mean sd n missing
## 23.26 24.12 24.6 24.99 26.32 24.58207 0.6544975 29 2

with(WomenHeptathlon2016, stem(x200m))

##
## The decimal point is at the |
##
## 23 | 3
## 23 | 589
## 24 | 011123334
## 24 | 5667789
## 25 | 00112444
## 25 |
## 26 | 3

with(WomenHeptathlon2016, stem(long_jump))

##
## The decimal point is 1 digit(s) to the left of the |
##
## 54 | 1
## 56 | 2
## 58 | 181
## 60 | 0588002569
## 62 | 023501145
## 64 | 38158
```

Section 5.1: Using the Standard Deviation to Standardize Values

```
filter(WomenHeptathlon2016, last_name == "Thiam") %>%
  data.frame()

## first_name last_name x200m long_jump x800m high_jump x100m_hurdles
## 1 Nafissatou Thiam 25.1 6.58 136.54 1.98 13.56
## javelin shot_put
## 1 53.13 14.91

# calculate z-score with mean and sd from favstats
(6.58 - 6.17)/.247 # long jump

## [1] 1.659919

filter(WomenHeptathlon2016, last_name == "Johnson-Thompson") %>%
  data.frame()

## first_name last_name x200m long_jump x800m high_jump
## 1 Katarina Johnson-Thompson 23.26 6.51 130.47 1.98
## x100m_hurdles javelin shot_put
## 1 13.48 36.36 11.68

data.frame() converts an object into a data frame.
```

Section 5.2: Shifting and Scaling

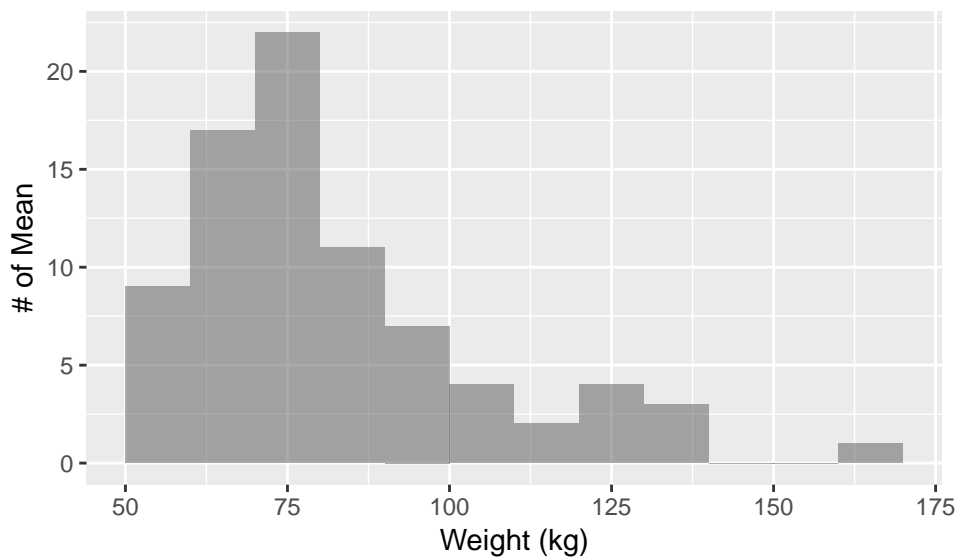
Shifting to Adjust the Center

```
MenWeight <- read_csv("http://nhorton.people.amherst.edu/is5/data/Mens_Weights.csv") %>%  
  clean_names()
```

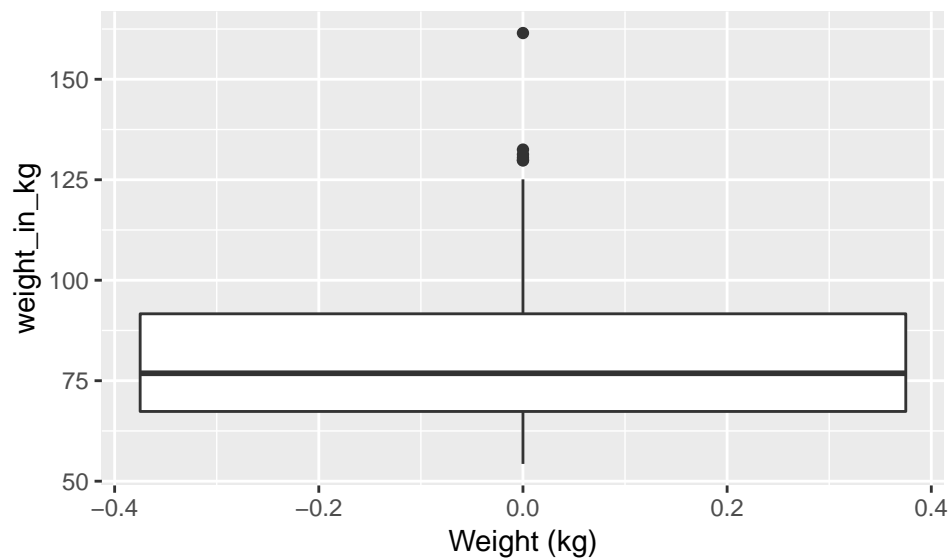
```
## Parsed with column specification:  
## cols(  
##   `Weight in kg` = col_double(),  
##   `Weight in pounds` = col_double()  
## )
```

Figure 5.2, page 125

```
gf_histogram(~ weight_in_kg, data = MenWeight, binwidth = 10, center = 5) %>%  
  gf_labs(x = "Weight (kg)", y = "# of Mean")
```



```
gf_boxplot(~ weight_in_kg, data = MenWeight, xlab = "Weight (kg)")
```

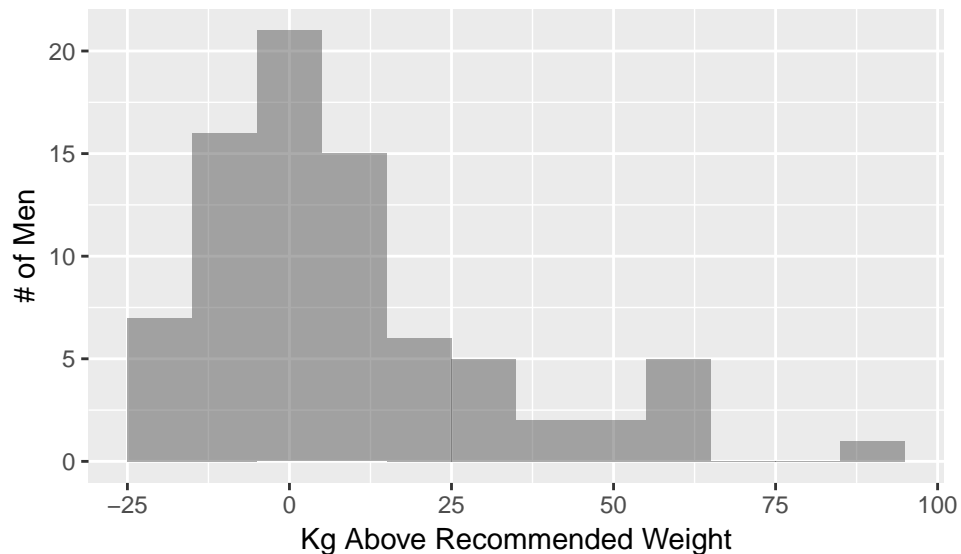


```
favstats(~ weight_in_kg, data = MenWeight)
```

```
##   min    Q1 median    Q3   max    mean      sd  n missing
##  54.3 67.35 76.85 91.65 161.5 82.35625 22.26881 80      0
```

Figure 5.3

```
gf_histogram(~ (weight_in_kg - 74), data = MenWeight, binwidth = 10) %>%
  gf_labs(x = "Kg Above Recommended Weight", y = "# of Men")
```



Rescaling to Adjust the Scale

```
favstats(~ weight_in_kg, data = MenWeight)
```

```
##   min    Q1 median    Q3   max    mean      sd  n missing
##  54.3 67.35 76.85 91.65 161.5 82.35625 22.26881 80      0
```

```
favstats(~ weight_in_pounds, data = MenWeight)
```

```
##   min    Q1 median    Q3   max    mean      sd  n missing
## 119.46 148.17 169.07 201.63 355.3 181.1838 48.99137 80      0
```

```
library(tidyr) # for gather() function
```

What does gather() do?

```
MenWeight %>%
```

```
  head() # There are two variables: weight_in_kg and weight_in_pounds. Each observation has a value for
```

```
## # A tibble: 6 x 2
```

```
##   weight_in_kg weight_in_pounds
```

```
##           <dbl>           <dbl>
```

```
## 1         107.           236.
```

```
## 2          95.7           211.
```

```
## 3          68.9           152.
```

```
## 4          60.3           133.
```

```
## 5          60.4           133.
```

```
## 6          69.7           153.
```

```
nrow(MenWeight)
```

```
## [1] 80
```

```
MenGather <- MenWeight %>%
```

```
  gather(key = weighttype, value = weight, weight_in_kg, weight_in_pounds)
```

```
MenGather %>%
```

```
  head() # The two variables are weighttype and weight. weighttype is a categorical variable that is ei
```

```
## # A tibble: 6 x 2
```

```
##   weighttype  weight
```

```
##   <chr>      <dbl>
```

```
## 1 weight_in_kg 107.
```

```
## 2 weight_in_kg 95.7
```

```
## 3 weight_in_kg 68.9
```

```
## 4 weight_in_kg 60.3
```

```
## 5 weight_in_kg 60.4
```

```
## 6 weight_in_kg 69.7
```

```
nrow(MenGather) # Each observation from before is now two rows
```

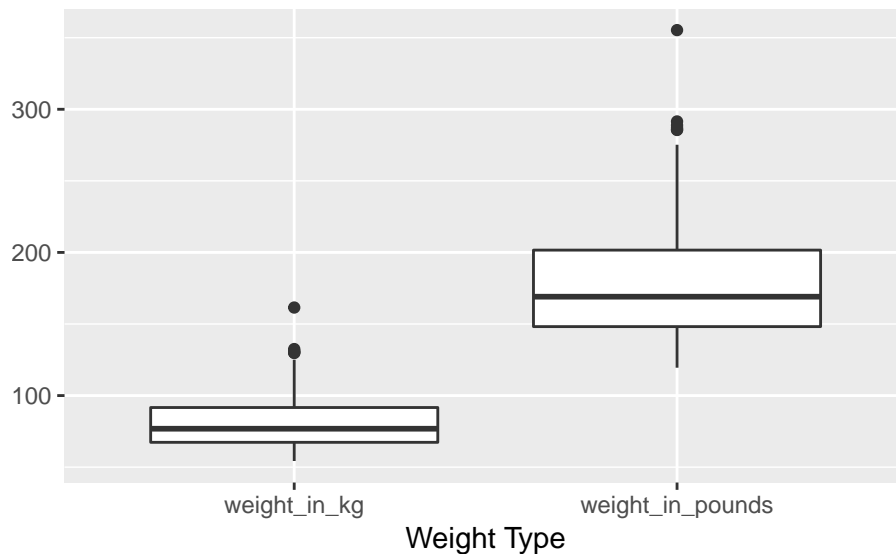
```
## [1] 160
```

Here we use the `gather()` function to transform the dataset into the needed format, which can be seen with the `head()` function.

```
MenGather %>%
```

```
  gf_boxplot(weight ~ weighttype) %>%
```

```
  gf_labs(x = "Weight Type", y = "")
```



We see the use of `goal(Y ~ X)` as an example of the general modeling language for two variables in the `mosaic` package.

Shifting, Scaling, and the z-Scores

Section 5.3: Normal Models

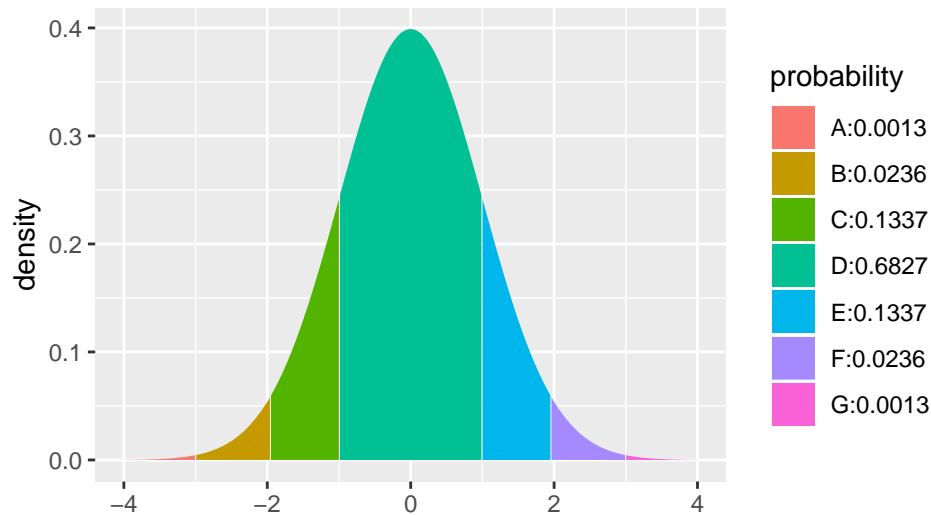
The 68-95-99.7 Rule

See display on page 129.

Figure 5.6

1, 2 (1.96), and 3 SD's

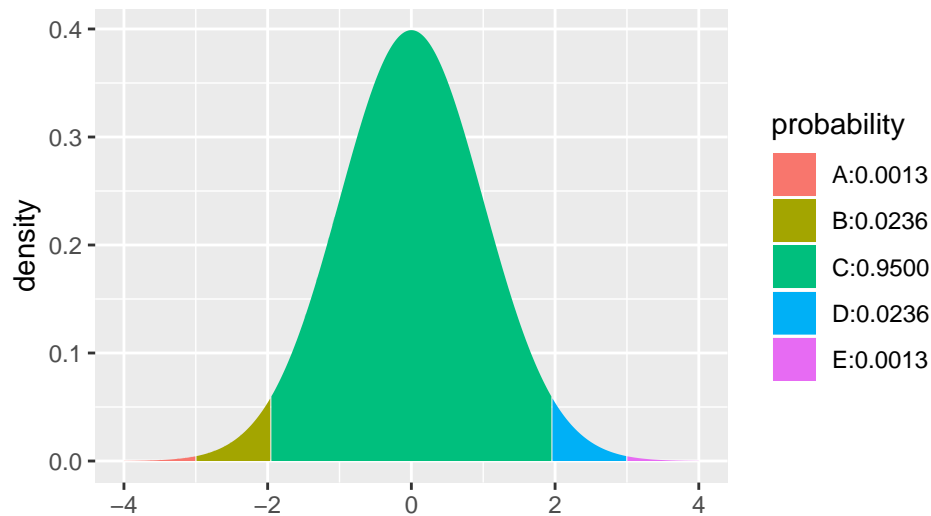
```
xpnorm(c(-3, -1.96, -1, 1, 1.96, 3), mean = 0, sd = 1, verbose = FALSE)
```



```
## [1] 0.001349898 0.024997895 0.158655254 0.841344746 0.975002105 0.998650102
```

2 (1.96) and 3 SD's

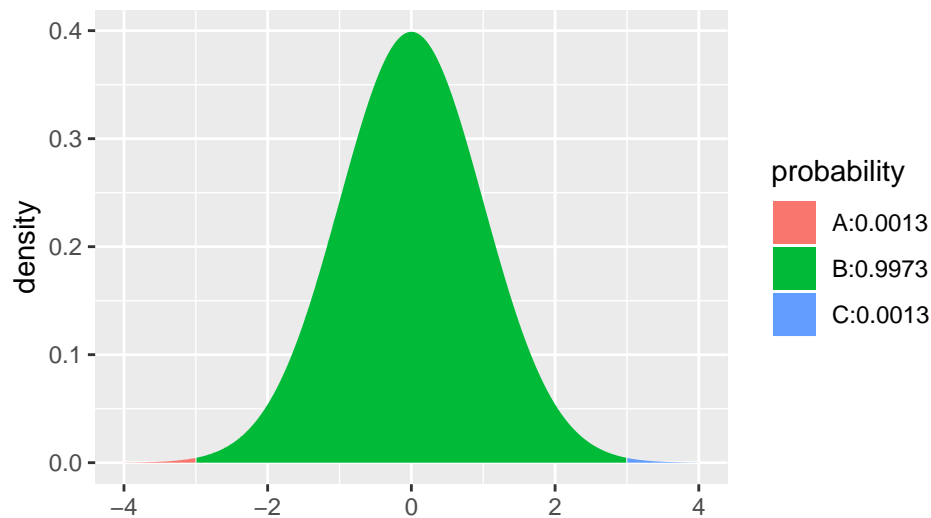
```
xpnorm(c(-3, -1.96, 1.96, 3), mean = 0, sd = 1, verbose = FALSE)
```



```
## [1] 0.001349898 0.024997895 0.975002105 0.998650102
```

3 SD's

```
xpnorm(c(-3, 3), mean = 0, sd = 1, verbose = FALSE)
```



```
## [1] 0.001349898 0.998650102
```

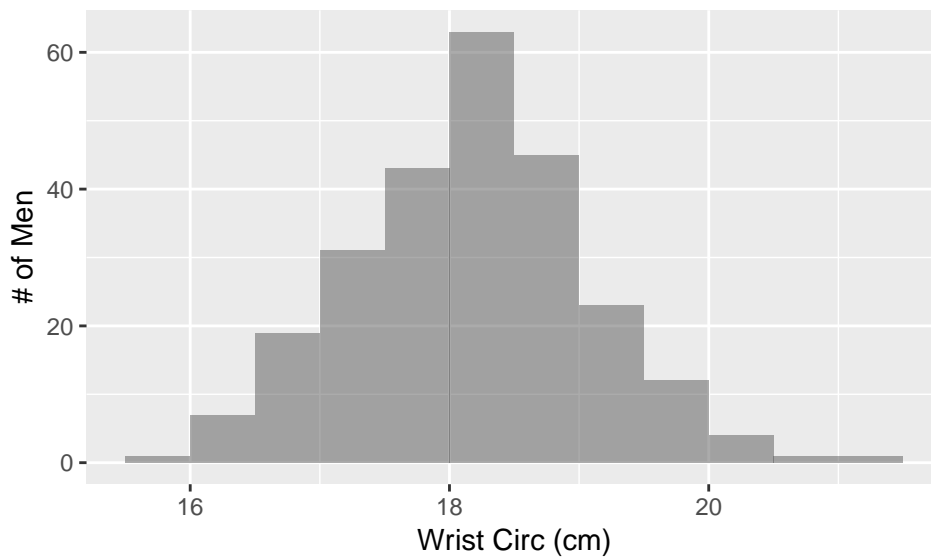
Example 5.4: Using the 68-95-99.7 Rule

```
BodyFat <- read_csv("http://nhorton.people.amherst.edu/is5/data/Bodyfat.csv")
```

```
## Parsed with column specification:
```

```
## cols(
##   Density = col_double(),
##   Pct.BF = col_double(),
##   Age = col_integer(),
##   Weight = col_double(),
##   Height = col_double(),
##   Neck = col_double(),
##   Chest = col_double(),
##   Abdomen = col_double(),
##   Waist = col_double(),
##   Hip = col_double(),
##   Thigh = col_double(),
##   Knee = col_double(),
##   Ankle = col_double(),
##   Bicep = col_double(),
##   Forearm = col_double(),
##   Wrist = col_double()
## )
```

```
gf_histogram(~ Wrist, data = BodyFat, binwidth = .5,
             center = -.25) %>%
  gf_labs(x = "Wrist Circ (cm)", y = "# of Men")
```



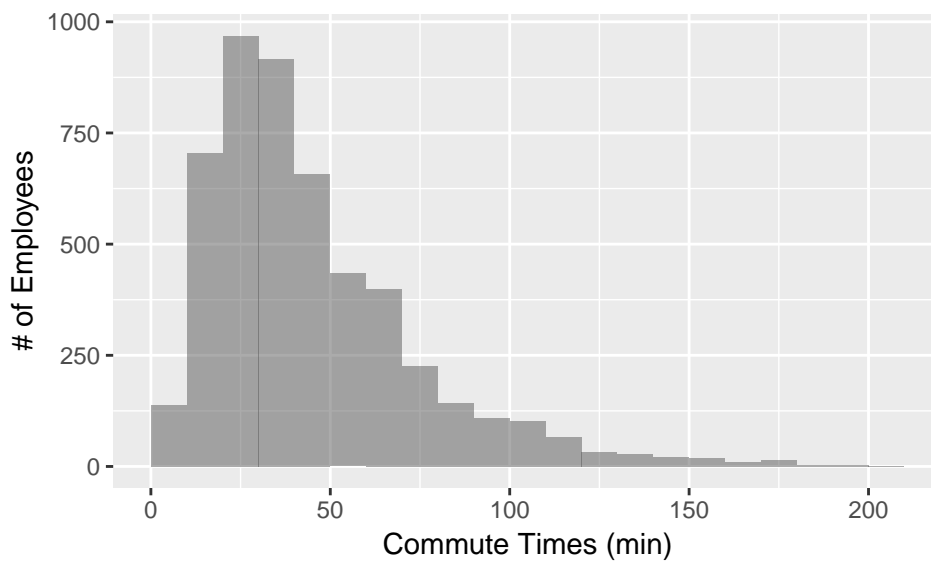
Random Matters

Starts on page 133.

```
Commute <-  
  read_csv("http://nhorton.people.amherst.edu/is5/data/Population_Commute_Times.csv") %>%  
  clean_names()
```

```
## Parsed with column specification:  
## cols(  
##   Commute.Time = col_integer()  
## )
```

```
gf_histogram(~ commute_time, data = Commute, binwidth = 10, center = 5) %>%  
  gf_labs(x = "Commute Times (min)", y = "# of Employees")
```



```
set.seed(2143) # To ensure we get the same values when we run it multiple times  
numsim <- 10000 # Number of simulations
```



```
mean(~ commute_time, data = sample(Commute, size = 100)) # Mean of one random sample

## [1] 46.77

mean(~ commute_time, data = sample(Commute, size = 100)) # Mean of another random sample

## [1] 42.07
```

The `do()` command allows us to run a command multiple times, saving the result as a data frame.

```
do(2) * mean(~ commute_time, data = sample(Commute, size = 100))

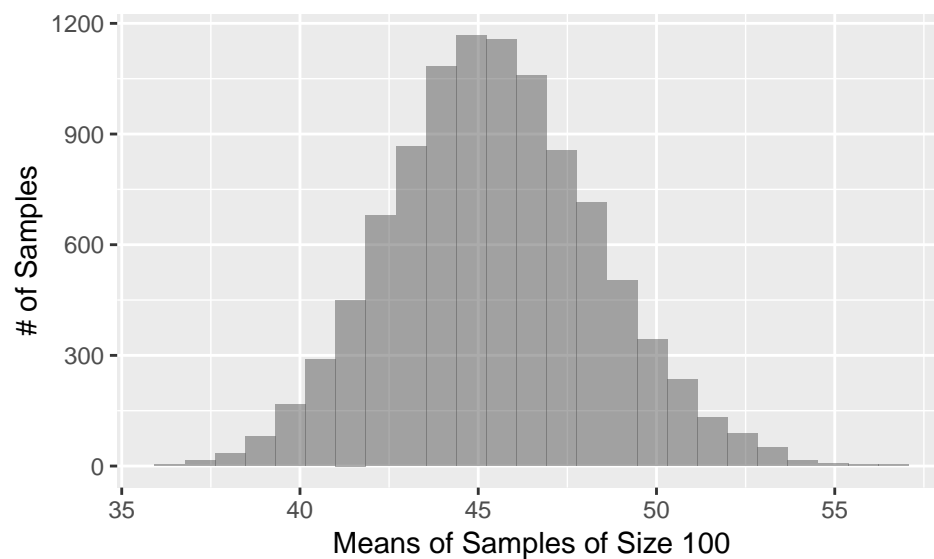
##      mean
## 1 42.19
## 2 42.90
```

For the visualization, we use do() 10,000 times

```
Commute_sample <- do(numsim) * mean(~ commute_time, data = sample(Commute, size = 100))
```

The `do()` function generates 10,000 samples of size 100 and for each calculates the sample mean.

```
gf_histogram(~ mean, data = Commute_sample) %>%
  gf_labs(x = "Means of Samples of Size 100", y = "# of Samples")
```

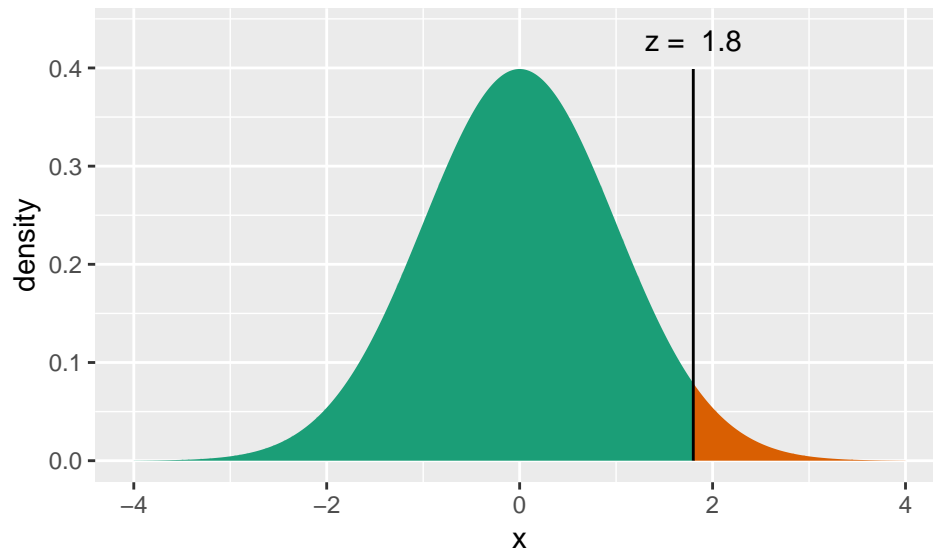


Section 5.4: Working with Normal Percentiles

The `pnorm()` function calculates normal probabilities. The `xpnorm()` function from the `mosaic` package adds a graphical depiction and additional output that may be helpful to new users.

```
xpnorm(1.8, mean = 0, sd = 1)

##
## If  $X \sim N(0, 1)$ , then
##  $P(X \leq 1.8) = P(Z \leq 1.8) = 0.9641$ 
##  $P(X > 1.8) = P(Z > 1.8) = 0.03593$ 
##
```



```
## [1] 0.9640697
```

The `qnorm()` function finds the inverse of normal probabilities.

```
xqnorm(0.964, mean = 500, sd = 100) # inverse of pnorm()
```

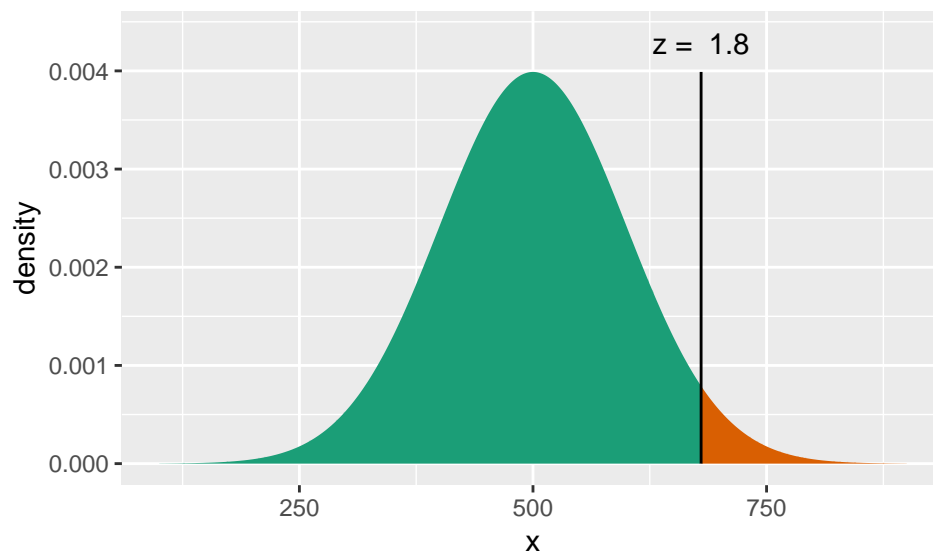
```
##
```

```
## If  $X \sim N(500, 100)$ , then
```

```
##  $P(X \leq 679.9118) = 0.964$ 
```

```
##  $P(X > 679.9118) = 0.036$ 
```

```
##
```



```
## [1] 679.9118
```

```
qnorm(0.964, mean = 0, sd = 1) # what is the z-score?
```

```
## [1] 1.799118
```

See examples on pages 136-140.

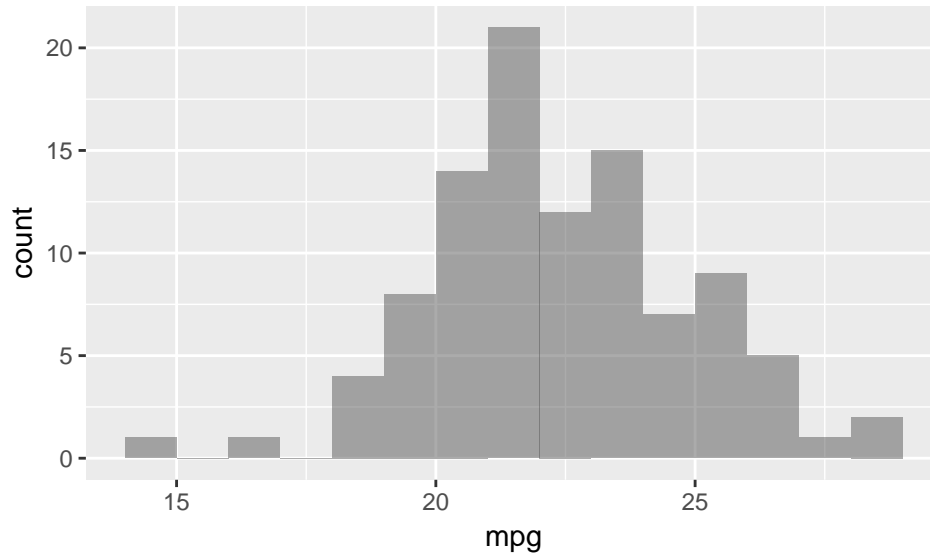
Section 5.5: Normal Probability Plots

```
Nissan <- read_csv("http://nhorton.people.amherst.edu/is5/data/Nissan.csv")
```

```
## Parsed with column specification:  
## cols(  
##   mpg = col_double()  
## )
```

Figure 5.10, page 141

```
gf_histogram(~ mpg, data = Nissan, binwidth = 1, center = .5)
```



```
gf_qq(~ mpg, data = Nissan, xlab = "Normal Scores") %>%  
  gf_qqline(linetype = "solid", color = "red")
```

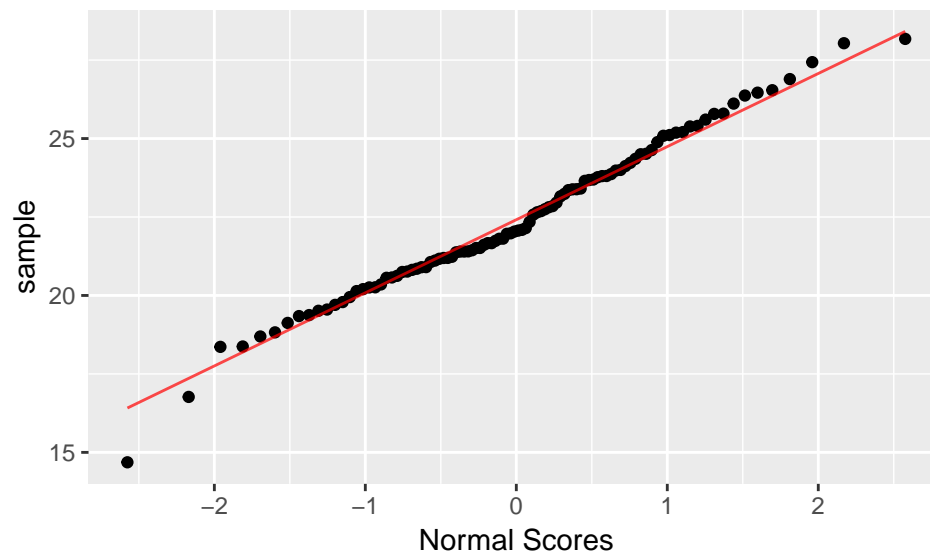
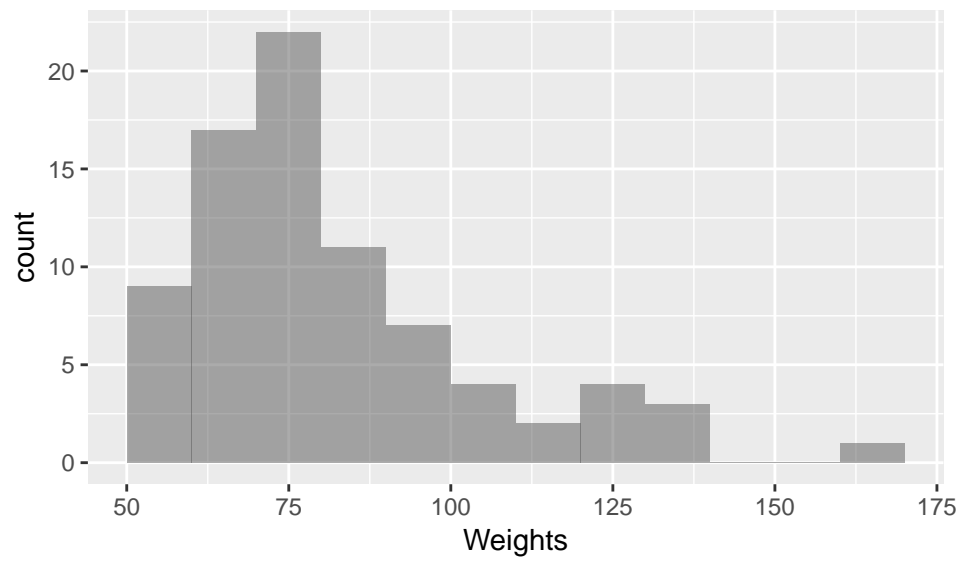


Figure 5.11

```
gf_histogram(~ weight_in_kg, data = MenWeight, xlab = "Weights", binwidth = 10, center = 5)
```



```
gf_qq(~ weight_in_kg, data = MenWeight, xlab = "Normal Scores") %>%
  gf_qqline(linetype = "solid", color = "red")
```

