IPS9 in R: Looking at Data – Distributions (Chapter 1)

Bonnie Lin and Nicholas Horton (nhorton@amherst.edu)

July 12, 2018

Introduction and background

This document is intended to help describe how to undertake analyses introduced as examples in the Ninth Edition of *Introduction to the Practice of Statistics* (2017) by Moore, McCabe, and Craig.

More information about the book can be found at https://macmillanlearning.com/Catalog/product/introductiontothepracticeofstatistics-ninthedition-moore. This file as well as the associated R Markdown reproducible analysis source file used to create it can be found at https://nhorton.people.amherst.edu/ips9/.

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 1: Looking at data – distributions

The specific goal of this document is to demonstrate how to replicate the analysis described in Chapter 1: Looking at Data (Distributions).

First, load the packages that will be needed for this document:

```
library(mosaic)
library(readr)
library(janitor)
```

Section 1.1: Data

Section 1.2: Displaying distributions with graphs

The table on page 9 displays the counts of preferences for online resources of 552 first-year college students. We begin by reading the data:

```
Online <-
  read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter01/EG01-07ONLINE.csv")
```

```
## Parsed with column specification:
## cols(
##   Source = col_character(),
##   Count = col_integer()
## )
```

```
Online %>%
  adorn_totals("row")
```

```
##     Source Count
##     Google   406
```

```
##    Library    75
##  Wikipedia    52
##      Other    19
##      Total   552
```

By default, the `read_csv()` function will output the types of columns, as we see above. To improve readability for future coding, we will suppress the "Parsed with column specification" message by adding `message = FALSE` at the top of the code chunks.

We can represent the data in percentages by dividing the number of students who favor each online resource by the total number of participants and multiplying the ratio by 100. The table on page 10 shows the preference percents.

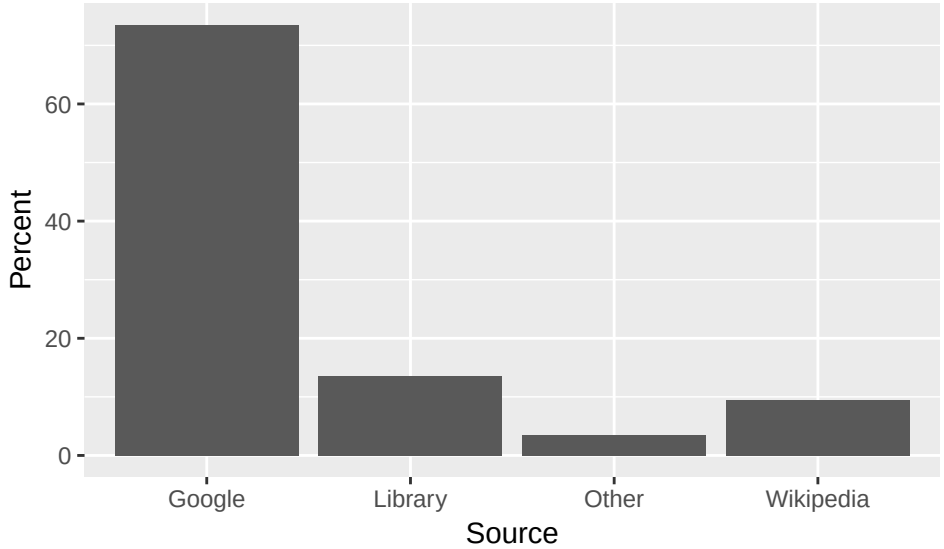
```
Online_Percent <-
  read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter01/EG01-08ONLINE.csv")

Online_Percent <- Online %>%
  mutate(Count = 100 * Count/sum(Count)) %>%
  rename(Percent = Count)

Online_Percent %>%
  adorn_totals("row")
```

```
##     Source    Percent
##     Google  73.550725
##    Library  13.586957
##  Wikipedia   9.420290
##      Other   3.442029
##      Total 100.000000
```

We use the `mutate()` function to compute the counts as percentages, while the `rename()` function provides an easy way to rename the column from "Count" to "Percent".

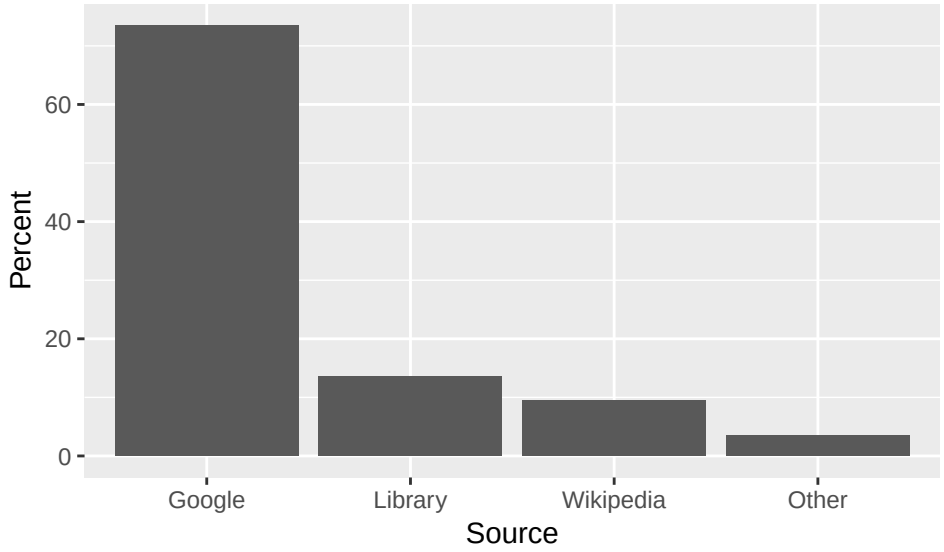
Figure 1.2 (page 10) displays the online resource preference data from the above example using a bar graph. We can make a bar graph by typing:

```
gf_col(Percent ~ Source, data = Online_Percent)
```




```
Online_Percent %>%
  arrange(-Percent) %>%
```

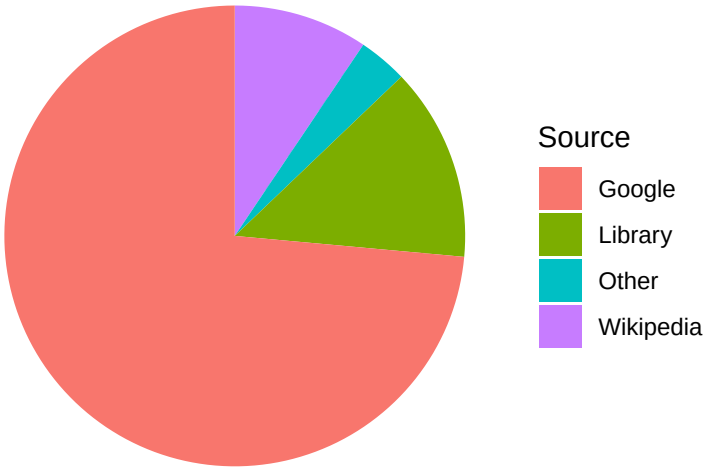
```
  mutate(Source = reorder(Source, - Percent)) %>%
  gf_col(Percent ~ Source)
```



R automatically orders the x-axis alphabetically, placing "Other" before "Wikipedia". However, we can make slight modifications by nesting the `reorder()` function in the `mutate()` function, which will first reorder the data based on the Percent and then reassign the data. The output now matches the graph on page 10 and graphs the sources on a descending order of preference percentages.

Figure 1.3 (page 11) displays the online resource preference data in a pie chart. You can create a simple one using the `ggplot2` package, which is called with the `mosaic` package. However, it is worth noting that the R documentation does not generally recommend pie charts as their features are somewhat limited.

```
Preferences <- ggplot(Online_Percent, aes(x = "", y = Percent, fill = factor(Source))) +
geom_bar(width = 1, stat = "identity")
Preferences + coord_polar(theta="y") + labs(fill = "Source") +  theme_void()
```



Below is the code that generates the stem and leaf plot (page 12) that displays the effect of SCF on the absorption of calcium in adolescent boys and girls:

```
SCF <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter01/EG01-11SCF.csv")
SCF_Treatment <- SCF %>%
  filter(Treatment == "SCF")
```

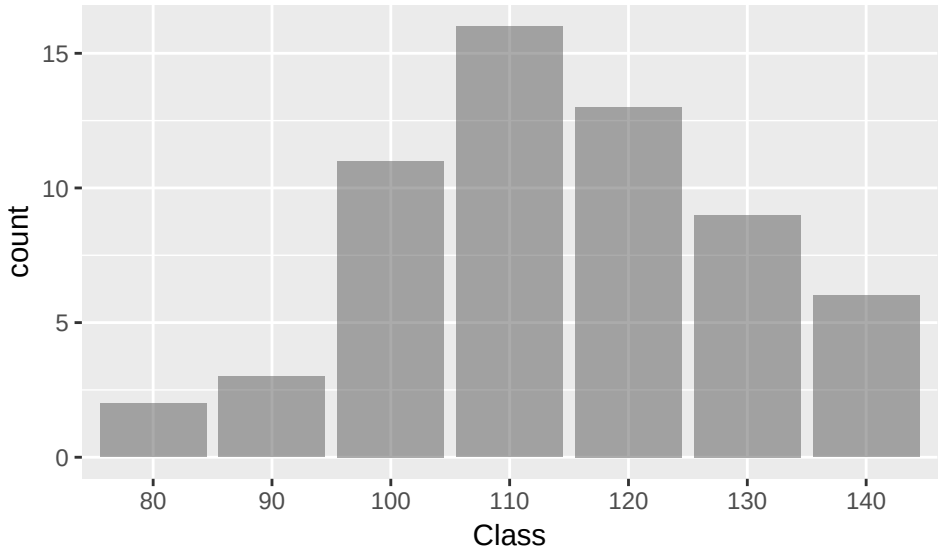
```
with(data = SCF_Treatment, expr = stem(Absorption))
```

```
##
##   The decimal point is 1 digit(s) to the right of the |
##
##   3 | 15
##   4 | 2333445789
##   5 | 003349
##   6 | 12
##   7 | 036
```

Since the data include absorption percentages of the control participants, we use the `filter()` function to isolate the boys and girls who consumed 12 grams per day of SCF from the control group. Once we have defined this subgroup, we can create a stemplot with the `with()` function, in which the arguments require the data and the expression `stem()`. This calls the variable `Absorption` within the `SCF_Treatment` data is and displays the vector as a stem-and-leaf plot.

Figure 1.7 (page 15) shows the IQ scores of 60 fifth-grade students chosen at random from one school. The data first need to be divided into classes or levels of equal width. We can then count the number of individuals in each class/level and use these counts to create a histogram.

```
IQ <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter01/EG01-14IQ.csv")
levels <- c(75, 85, 95, 105, 115, 125, 135, 145, 155)
labels <- c("75 <= IQ & IQ < 85", "85 <= IQ & IQ < 95", "95 <= IQ & IQ < 105", "105 <= IQ & IQ < 115",

labels <- as.factor(seq(from = 80, to = 150, by = 10))

IQ_Count <- IQ %>%
  mutate(Class = cut(IQ, levels, labels = labels))
gf_histogram(~ Class, data = IQ_Count, stat = "count")
```




```
## XX Thoughts on which one we prefer?
```

The `cut()` function divides the IQ dataset into the defined levels and assigns the values into the appropriate categories.
By setting the `stat` argument in the `gf_histogram()` function to "count", we can create a histogram for discrete variables. The distribution of scores on IQ tests is roughly "bell-shaped".

Table 1.2 (page 17) displays the lengths of the first 80 calls. To call the first 80 observations in the dataset,

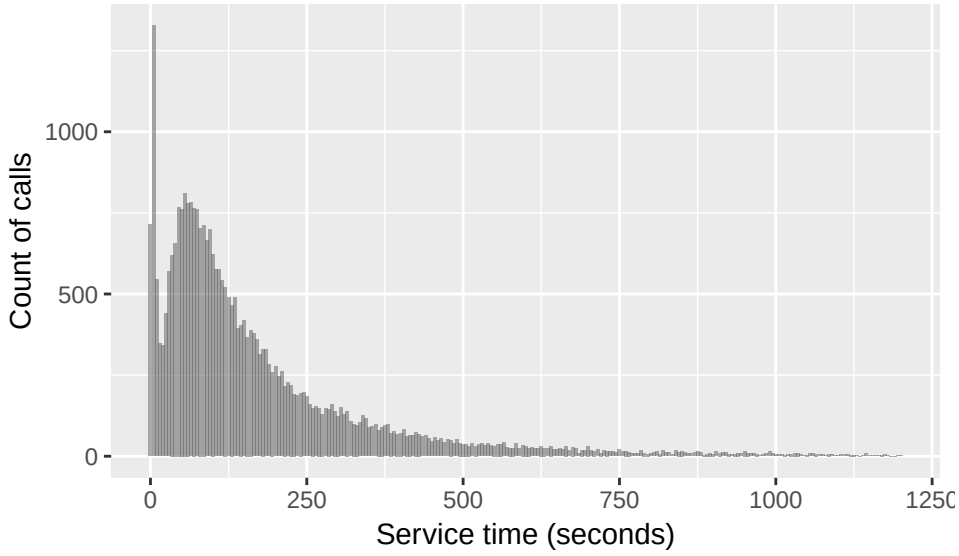
we can use the `head()` function and specify the number of observations.

```
Customer_Calls <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter01/EG01-16CALLS.csv")
head(Customer_Calls, 80)
```

```
## # A tibble: 80 x 1
##    length
##     <int>
##  1     77
##  2    289
##  3    128
##  4     59
##  5     19
##  6    148
##  7    157
##  8    203
##  9    126
## 10    118
## # ... with 70 more rows
```

Figure 1.8 displays the histogram of the lengths of all 31,492 calls. To exclude the few lengths that are greater than 1200 seconds (20 minutes), we can use the `filter()` function.

```
Customer_Calls <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter01/EG01-16CALLS.csv")
Customer_Calls %>%
  filter(length <= 1200) %>%
  gf_histogram(~ length, binwidth = 5) %>%
  gf_labs(x = "Service time (seconds)", y = "Count of calls")
```



From the last visual, we can assess the shape of the distribution of IQ scores. We can also use the `favstats()` function to find the center and spread:

```
favstats(~ IQ, data = IQ)
```

```
##  min    Q1 median     Q3 max     mean       sd  n missing
##   81 104.5    114 125.25 145 114.9833 14.80093 60       0
```

For the exact center, we can see that the median IQ score is 114. As for the measure of spread, we can use the minimum value and the maximum value to say that the spread is from 81 to 145. Both the distribution

as well as the summary statistics point to a symmetric, unimodal pattern in the IQ data.

On the other hand, the distribution of call lengths displayed above is strongly *skewed to the right.* We can run the `favstats()` function to find exact values of the center and spread:

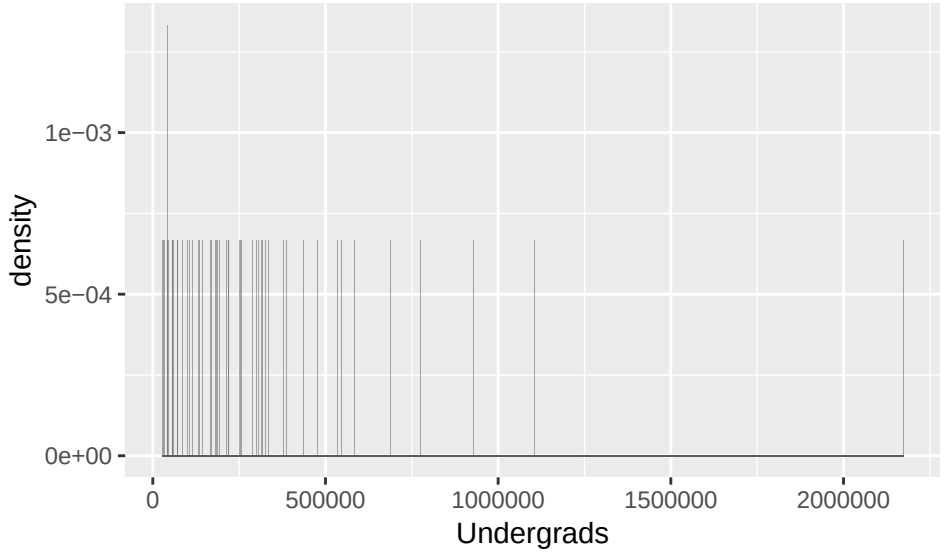
```
favstats(~ length, data = Customer_Calls)
```

```
##  min Q1 median  Q3   max     mean       sd     n missing
##    1 57    115 225 28739 188.5922 312.7768 31492       0
```

We can say that the length of a typical call is about the median value of the dataset, or 115 seconds. The data spread very widely, spanning from 1 second to 28, 739 seconds. The outliers were omitted from the histogram above, but it is still clear thats the shape of the distribution is hardly symmetric.

Next, we will look at the number of undergraduate college students in each of the states. Here, we have loaded in the data and displayed the data in a histogram. See Figure 1.9 (page 20):

```
College <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter01/EG01-19COLLEGE.csv")

gf_dhistogram(~ Undergrads, data = College, binwidth = 30)
```




```
College %>%
  filter(Undergrads == max(Undergrads))
```

```
## # A tibble: 1 x 4
##   State      Undergrads Population UGradPerThou
##   <chr>           <int>      <int>        <dbl>
## 1 California    2172354   36121296         60.1
```

```
## XX Comes out not looking so pretty in the PDF and not sure why the density plot looks
## that way
```

California is the outlier in this case, by having 2,687,893 undergraduate students.

The *UGradPerThou* variable in the dataset takes into account the variation in state populations and expresses the number of undergraduate students per 1000 people in each state. Below is a stemplot, which provides essentially the same information as a histogram with the added benefit of being able to easily extract indiviudal data points.

```
with(College, stem(UGradPerThou))
```

```
##
##   The decimal point is 1 digit(s) to the right of the |
##
##   3 | 8
##   4 | 1111133
##   4 | 55556667777788889
##   5 | 001112224444
##   5 | 566
##   6 | 00001
##   6 | 79
##   7 | 12
##   7 | 7
```

```
College %>%
  filter(UGradPerThou > 76)
```

```
## # A tibble: 1 x 4
##   State   Undergrads Population UGradPerThou
##   <chr>        <int>      <int>        <dbl>
## 1 Arizona     476547    6178251         77.1
```

To interpret this plot alongside the histogram, we can take California for example. California has 60 undergraduate students per 1000 people, making it one of the higher values, though certainly not the highest. Though the last histogram established California as an outlier for its extremely large number of undergraduate students, the stemplot points out another state as an outlier for having the largest proportion of undergraduate students.

Since we can view individual points in stemplots, we can use the `filter()` function to find the name of the outlier state, which turns out to be Arizona, with a ratio of 77.1 undergraduate students per 1000 people in the state.

Below is the code that we use to create a stemplot of the values of PTH measured on a sample of 29 boys and girls aged 12 to 15 years (Figure 1.11 on page 21):

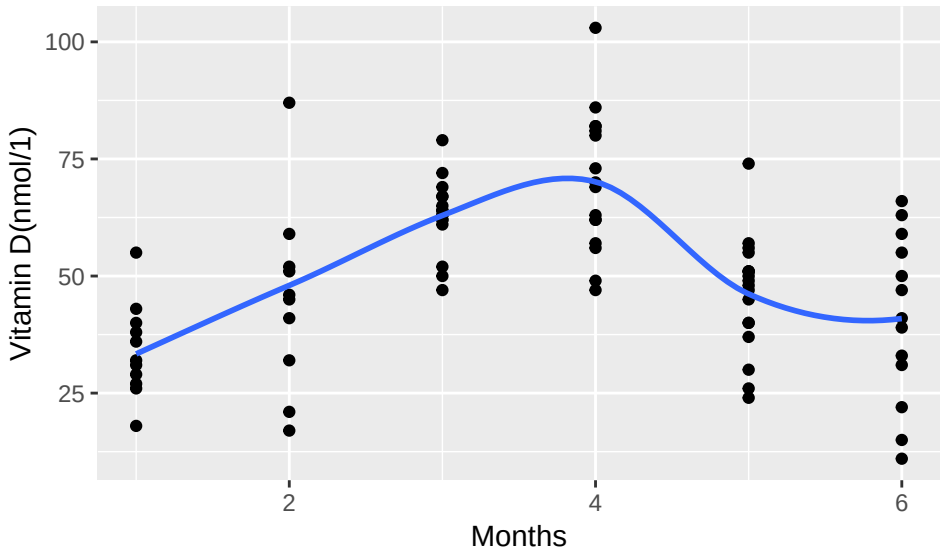
```
PTH <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter01/EG01-21PTH.csv")
with(PTH, stem(PTH))
```

```
##
##   The decimal point is 1 digit(s) to the right of the |
##
##    0 | 9
##    2 | 5888901113589
##    4 | 0568990099
##    6 | 3411
##    8 |
##   10 |
##   12 | 7
```

The 127 PTH value at the base of the stemplot stands out as the outlier.

Following is an example of a plot that displays observations in time order. The serum levels of vitamin have been plotted against time of year for samples of subjects from Switzerland (see Figure 1.12 on page 22).

```
VITDS <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter01/EG01-22VITDS.csv")
gf_point(VitaminD ~ Months, data = VITDS) %>%
  gf_smooth(se = FALSE) %>%
  gf_labs(x = "Months", y= "Vitamin D(nmol/1)")
```



By default, the `se` argument in `gf_smooth()` function is set to TRUE, so we set it to FALSE to suppress the confidence intervals.

Section 1.3: Describing Distributions with Numbers

Like the previous examples of stemplots, you can type the following to display the data collected on the time, in days, that businesses took to complete all of the starting procedures (page 29):

```
TTS24 <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter01/EG01-23TTS24.csv")
with(TTS24, stem(Time, scale = 2))
```

```
##
##   The decimal point is 1 digit(s) to the right of the |
##
##   0 | 2455556678
##   1 | 01236799
##   2 | 45
##   3 | 28
##   4 | 9
##   5 | 3
```

We can control the length of the stemplot by changing the `scale` argument from 1, which is the default setting, to 2.

Other than using the `favstats()` function to calculate summary statistics, we can also use the `mean()` function to find this specific statistic.

```
mean(~ Time, data = TTS24)
```

```
## [1] 16.29167
```

XX ?? use this options(digits =) or round()

```
Suricane <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter01/EG01-25TTS24.csv")
mean(~ Time, data = Suricane) %>% round(digits = 2)
```

```
## [1] 16.29
```

We have piped the `mean()` output into the `round()` function and specified the number of digits after the decimal to be 2, as the textbook instructs on page 29.

Similarly, we can use the `median()` function to find this specific statistic:

```
median(~ Time, data = TTS24)
```

```
## [1] 11.5
```

You can also use the stemplot generated above to confirm the median value of this variable.

Other functions that operate similarly include:

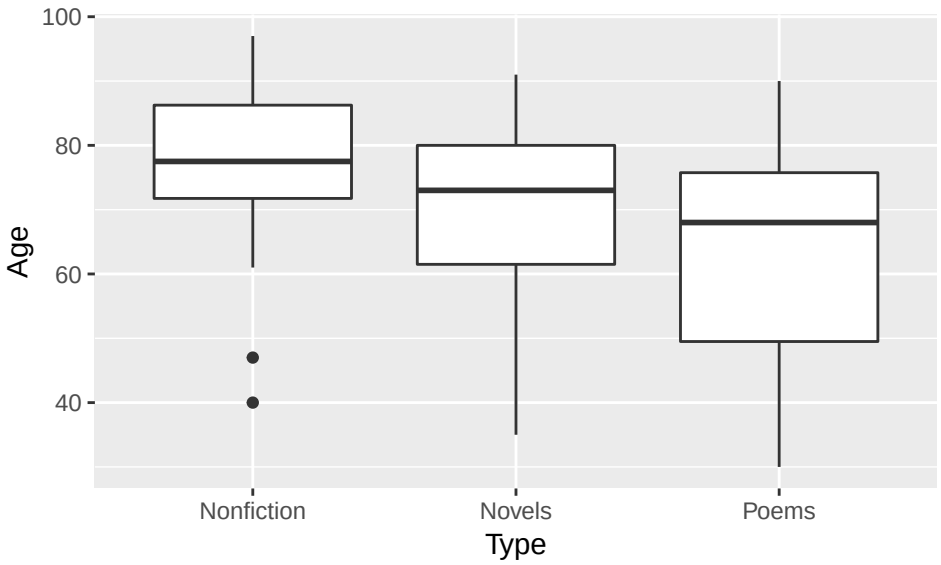
- range(): to find the maximum and minimum values
- IRQ(): to calculate the quartiles (Q3 - Q1)

You can type either line of code to get the help file for the function to see more information:

```
?favstats
favstats()
```

Figure 1.15 (page 37) displays side-by-side boxplots for age of the death of writers in the three categories. of literature.

```
POETS <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter01/EG01-31POETS.csv")
gf_boxplot(Age ~ Type, data = POETS)
```



To demonstrate how to linearly transform vectors, we will use the homework scores and the corresponding final grade points for five students (page 45):

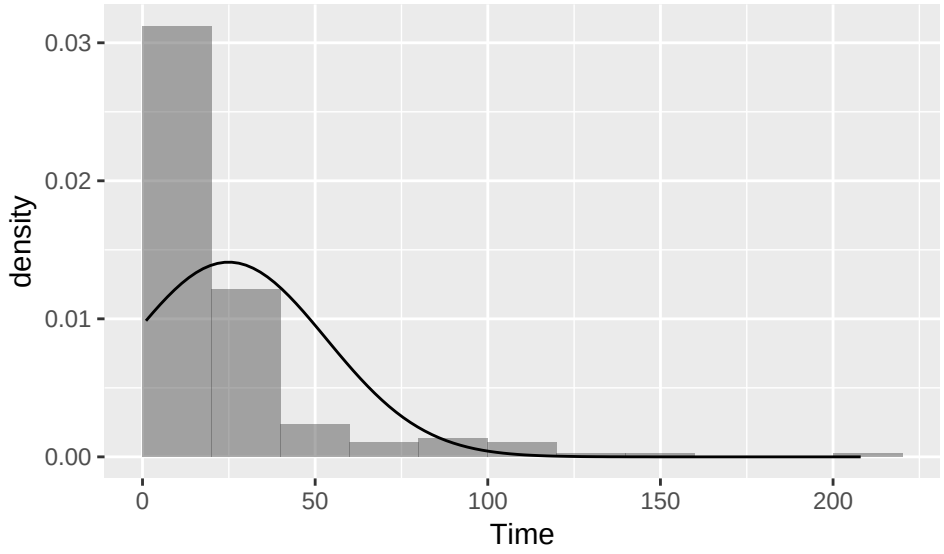
```
Score <- c(1056, 1080, 900, 1164, 1020)
Points <- Score/4
mean(Score)/mean(Points)
```

```
## [1] 4
```

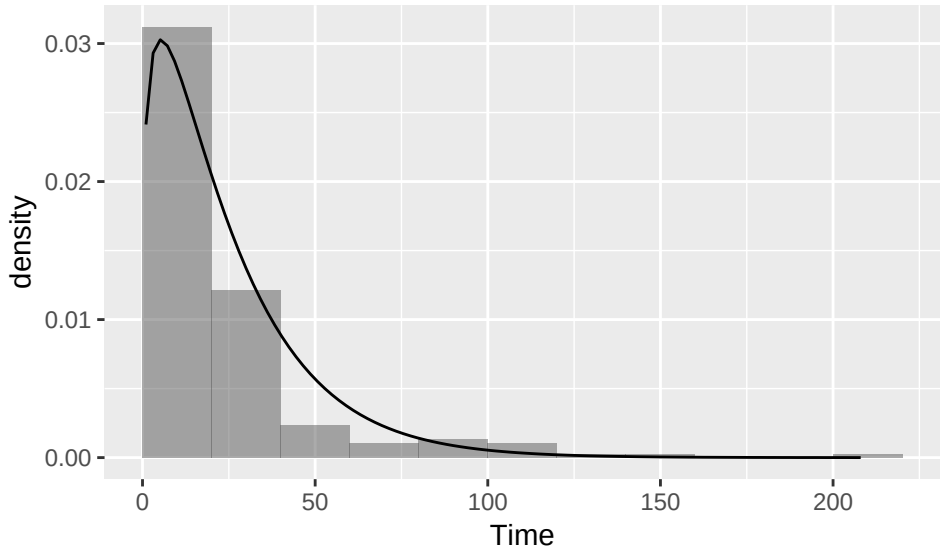
We create a vector called `Score` that holds the total score of the 12 homework assignments. We can convert the homework scores to final grade points by dividing each value in the `Score` vector by 4. Because this is a linear transformamtion, the mean of the summed homework score is just four times the mean of the points.

Figure 1.20(a) (page 52) shows the distribution of the time it takes to start a business, as a density plot overlain a histogram. We can recreate this by typing:

```
TTS <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter01/EG01-36TITANIC.csv")
gf_dhistogram(~ Time, data = TTS, binwidth = 20, center = 10) %>%
  gf_fitdistr()
```




```
gf_dhistogram( ~ Time, data = TTS, binwidth = 20, center = 10) %>%
  gf_fitdistr(dist = "dgamma")
```

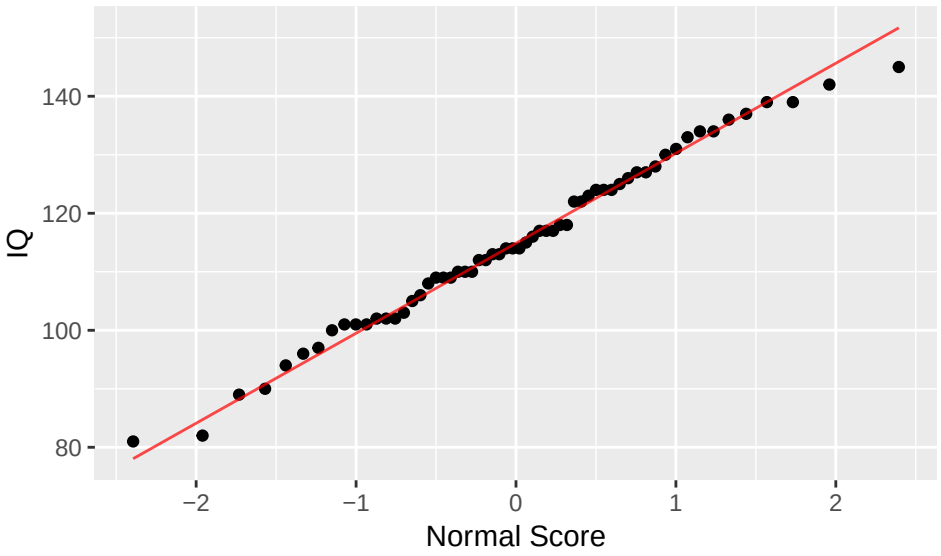



```
## XX Closest I can get to replicating the textbook is the second graph, but I'm not sure how to overlay
## the actual density onto the histogram
```

Section 1.4: Density Curves and Normal Distributions

Figure 1.30 (page 67) is a Normal quantile plot of the 60 fifth-grade IQ scores from page 14. We can recreate the plot here:

```
IQ <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter01/EG01-46IQ.csv")
gf_qq(~ IQ, data = IQ) %>%
  gf_qqline(linetype = "solid", color = "red") %>%
  gf_labs(x = "Normal Score", y = "IQ")
```



Since the points do not deviate far from the straight line drawn on the plot, we can conclude that the distribution of the IQ data is approximately Normal. Here, we use the `gf_qq()` function to plot the points and then add the `gf_qqline()` function, with the correct specifications, to overlay the straight diagonal line.