Impressions of R from an Introductory Course on Statistics

William John Holden

10 May 2020

```r
library(readxl)
library(ggplot2)
library(dplyr)
library(tidyr)
library(comprehenr)
library(knitr)
```

Introduction

Over the past eight weeks I took Introduction to Statistics (STAT 200) at the University of Maryland Global Campus (UMGC). My bachelor's program in computer science did not require a course in statistics and I felt this was something I needed before applying to graduate school.

UMGC teaches STAT200 from the second edition of Statistics Using Technology by Kathryn Kozak. UMGC does not require the use of any specific technology for this course. The textbook shows functions from a Texas Instruments calculator. You could probably get away with Microsoft Excel if you are very clever. I took this opportunity to learn R.

I would rate my own skill in computer programming somewhere above novice but below intermediate. My day job is in computer networking, where I do not write code every day. Most of my programs are in (in descending order of strength) Java, Julia, JavaScript, and Mathematica.

This document is a summary of my experience self-teaching R. I did not bother trying to learn R as a general-purpose programming language and I have no intention to use R for Advent of Code. I think of R as a domain-specific language for doing statistics. I would never consider R for building a web application, video game, desktop application, or network service.

RStudio

I used a program called RStudio. There is a no-cost, open-source version of RStudio Desktop. Before you install RStudio you will need the core R language, which you can download from CRAN.

RStudio is great. The core R language is a REPL. RStudio provides both a code view and console view that allow you to effortlessly switch from REPL to code. RStudio's graphical user interface provides view of your environment variables, which feels a little bit like a debugger in a standard IDE. RStudio prominently features a view for plots, which makes sense for a product aimed at statisticians.

R Markdown

R Markdown is a "killer app." R Markdown is a notebook interface where you mix prose, mathematical notation, code, and code output. RStudio can output to LaTeX. If you already know LaTeX then you can less R Markdown in about 2 minutes (which includes the 60 seconds to learn Markdown.

Cosma Shalizi has an introduction to R Markdown at Using R Markdown for Class Reports.

The other two notebook interfaces I have used are Mathematica and Jupyter. Mathematical notation is important to me. I hated the palettes in Mathematica. I never memorized the keyboard shortcuts and the graphical user interface was painfully slow and difficult to use. The equation editor in Microsoft Office is easier to use. I had no trouble learning Jupyter because I had already invested in learning LaTeX. I recent watched an excellent talk by Joel Grus called "I don't like notebooks". Grus' chief frustration with notebooks is hidden and confusing state, such as out-of-order evaluation. I think RStudio users might be less likely to encouter these problems than Jupyter users. An RStudio user is likely to try their computation at the console first, then save their history to an R Script when they are satisfied with their result. Using the environment as a debugger to look for a missing step should be a natural first step if the R Script does not work as expected.

R Markdown manages to put together a lot of things that worked in the industry. Prose, code, and plots are easily blended, just like Mathematica Jupyter notebooks. The source document is plain text and is readable in this form. Finally, you can use LaTeX to typeset beautiful math.

Reading Data

RStudio has a very useful Import Dataset feature. You can use this feature to navigate to a file, such as an Excel spreadsheet, containing your data, view the columns. The feature shows the R source code needed to produce the result you see on the screen. `readxls` is slick. For example:

```r
library(readxl)
CrossFit <- read_excel("CrossFit.xlsx")
kable(CrossFit)
```

Study.Authors	Year	x.Male	x.Female	x	n.Male	n.Female	n	Incidence
Hak, Hodzovic, & Hickey	2013	NA	NA	97	NA	NA	132	3.10
Giordano & Weisenthal	2014	NA	NA	NA	NA	NA	386	2.40
Weisenthal et al.	2014	53	21	75	231	150	386	NA
Chachula, Cameron, & Svoboda	2015	NA	NA	24	40	14	54	NA
Aune & Powers	2016	52	33	85	142	105	247	2.71
Sprey et al.	2016	109	67	176	323	243	566	NA
Summitt et al.	2016	NA	NA	44	NA	NA	187	1.94
Mehrab et al.	2017	157	95	252	266	183	449	NA
Montalvo et al.	2017	30	20	50	94	97	191	2.30
Moran et al.	2017	NA	NA	NA	66	51	117	2.10
Feito, Burrows, & Tabb	2018	495	436	931	1566	1483	3049	0.74
Tafuri et al.	2019	NA	NA	181	325	129	454	NA
da Costa et al.	2019	89	68	157	243	171	414	3.24
Minghelli & Vicente	2019	NA	NA	61	152	118	270	1.34
Alekseyev et al.	2020	198	97	295	589	296	885	NA
Larsen et al.	2020	NA	NA	25	51	117	168	10.60
Szeles et al.	2020	NA	NA	NA	198	208	406	18.90

The `kable` function helps format a data frame into a format that looks nice in LaTeX. In RStudio you can call `View` (with capital "V") or enter the name of your variable into the console and press enter.

Tidyverse

Getting data into any computing environment can be difficult. During Advent of Code 2019 I felt Julia made importing plaintext input fairly easy with its `readdlm`. I had a less positive experience of Mathematica's `Import` command in Advent of Code 2018. Granted, I did not use R for general-purpose computing.

Getting data into a usable format for R can be difficult and tedious. I recommend watching this four-video series called "Data Wrangling with R and the Tidyverse" by Garrett Grolemund:

1. Introduction to Data Wrangling
2. Tidy Data and `tidyr`
3. Data Manipulation Tools: `dplyr`
4. Working with Two Datasets: Binds, Set Operations, and Joins

Tidyverse refers to to a collection of R packages that work well together to manipulate and visualize data. The two packages that I mainly used in STAT200 were `dplyr` and `ggplot2`.

`%>%`

One of my favorite features in the Tidyverse is the `%>%` operator. This operator pipes input from the last command as an anonymous variable and supplies this anonymous variable as the first parameter of the next function. So, `g(f(x, y), z)` is semantically the same as `f(x, y) %>% g(z)`. This may look strange but it very easy to use in practice.

I really like pipelines in bash and Powershell. Let's compare the sum of the square roots of five random numbers using a pipeline in a few programming languages.

$$\sum_{i=1}^{5} \sqrt{\texttt{Random}[0,1]}$$

Java 8's Stream API can do this in a predictably verbose form.

```
DoubleStream.generate(Math::random).limit(5).map(Math::sqrt).sum()
```

JavaScript truly is a Lisp in C's clothing.

```
Array.from({length: 5}, Math.random).map(Math.sqrt).reduce((a,b) => a + b, 0)
```

Julia has a weird `|>` operator, but you have to explicitly state the name of variables passed to some functions. All of these do the same thing:

```
rand(5) |> x -> map(sqrt, x) |> sum
rand(5) |> x -> sqrt.(x) |> sum
sum(map(sqrt, rand(5)))
sum(sqrt.(rand(5)))
```

Julia's `sqrt` function is not *vectorized*. `sqrt` expects a single value; you can `map` or the dot syntax to apply a function element-wise against an array.

Here is the R version:

```r
sum(sqrt(runif(5)))
```

```
## [1] 3.198304
```

```r
runif(5) %>% sqrt() %>% sum()
```

```
## [1] 3.064623
```

Vectorized Functions

The `sqrt` function in R is built on an assumption that, if given a list, it should operate element-wise and return a list of the same length. Many of R's built-in functions are vectorized:

```r
1:10 + 0
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
1:10 * runif(10)
```

```
##  [1] 0.9666563 1.2549697 2.2737568 2.1937730 2.5378871 1.4670244 3.9905612
##  [8] 6.8578826 6.7414081 3.1413934
```

```r
exp(1:10) / 2^(1:10)
```

```
##  [1]  1.359141  1.847264  2.510692  3.412384  4.637911  6.303575  8.567447
##  [8] 11.644367 15.826336 21.510221
```

If one vector is longer than the other then R will wrap around and reuse values from the shorter vector. This is completely different from Python's Python's zip function.

```r
1:10 + 1:9 + 1:8
```

```
## Warning in 1:10 + 1:9: longer object length is not a multiple of shorter object
## length

## Warning in 1:10 + 1:9 + 1:8: longer object length is not a multiple of shorter
## object length

##  [1]  3  6  9 12 15 18 21 24 19 13
```

dplyr

Some of the functions that I found especially useful in `dplyr` are `arrange`, `filter`, `select`, `mutate`, and `summarize`. There are many more powerful and highly composable functions in `dplyr`, such as `pivot_longer` (`gather`) and `pivot_wider` (`spread`), but I did not use them as often.

arrange

Sort the studies in descending order by year, then alphabetically by the authors' names.

```r
CrossFit %>% arrange(-Year, Study.Authors) %>% kable()
```

Study.Authors	Year	x.Male	x.Female	x	n.Male	n.Female	n	Incidence
Alekseyev et al.	2020	198	97	295	589	296	885	NA
Larsen et al.	2020	NA	NA	25	51	117	168	10.60
Szeles et al.	2020	NA	NA	NA	198	208	406	18.90
da Costa et al.	2019	89	68	157	243	171	414	3.24
Minghelli & Vicente	2019	NA	NA	61	152	118	270	1.34
Tafuri et al.	2019	NA	NA	181	325	129	454	NA
Feito, Burrows, & Tabb	2018	495	436	931	1566	1483	3049	0.74
Mehrab et al.	2017	157	95	252	266	183	449	NA
Montalvo et al.	2017	30	20	50	94	97	191	2.30
Moran et al.	2017	NA	NA	NA	66	51	117	2.10
Aune & Powers	2016	52	33	85	142	105	247	2.71
Sprey et al.	2016	109	67	176	323	243	566	NA
Summitt et al.	2016	NA	NA	44	NA	NA	187	1.94
Chachula, Cameron, & Svoboda	2015	NA	NA	24	40	14	54	NA
Giordano & Weisenthal	2014	NA	NA	NA	NA	NA	386	2.40
Weisenthal et al.	2014	53	21	75	231	150	386	NA
Hak, Hodzovic, & Hickey	2013	NA	NA	97	NA	NA	132	3.10

`filter` and `select`

Show only the studies which reported injury incidence (injuries per 1000 hours of exposure), and show only this column and the authors.

```r
CrossFit %>% filter(!is.na(Incidence)) %>% select(Study.Authors,Incidence) %>% kable()
```

Study.Authors	Incidence
Hak, Hodzovic, & Hickey	3.10
Giordano & Weisenthal	2.40
Aune & Powers	2.71
Summitt et al.	1.94
Montalvo et al.	2.30
Moran et al.	2.10
Feito, Burrows, & Tabb	0.74
da Costa et al.	3.24
Minghelli & Vicente	1.34
Larsen et al.	10.60
Szeles et al.	18.90

In this situation, you could get the same result with the `drop_na` function from `tidyr`:

```r
CrossFit %>% select(Study.Authors, Incidence) %>% drop_na()
```

`mutate`

Compute injury prevalence (the proportion of athletes injured in a sample) by gender:

```r
CrossFit <- CrossFit %>% mutate(p.Male = x.Male / n.Male, p.Female = x.Female / n.Female,
                                p = x / n)
CrossFit %>% select(Study.Authors, Year, p.Male, p.Female, p) %>% kable()
```

Study.Authors	Year	p.Male	p.Female	p
Hak, Hodzovic, & Hickey	2013	NA	NA	0.7348485
Giordano & Weisenthal	2014	NA	NA	NA
Weisenthal et al.	2014	0.2294372	0.1400000	0.1943005
Chachula, Cameron, & Svoboda	2015	NA	NA	0.4444444
Aune & Powers	2016	0.3661972	0.3142857	0.3441296
Sprey et al.	2016	0.3374613	0.2757202	0.3109541
Summitt et al.	2016	NA	NA	0.2352941
Mehrab et al.	2017	0.5902256	0.5191257	0.5612472
Montalvo et al.	2017	0.3191489	0.2061856	0.2617801
Moran et al.	2017	NA	NA	NA
Feito, Burrows, & Tabb	2018	0.3160920	0.2939987	0.3053460
Tafuri et al.	2019	NA	NA	0.3986784
da Costa et al.	2019	0.3662551	0.3976608	0.3792271
Minghelli & Vicente	2019	NA	NA	0.2259259
Alekseyev et al.	2020	0.3361630	0.3277027	0.3333333
Larsen et al.	2020	NA	NA	0.1488095
Szeles et al.	2020	NA	NA	NA

As shown, R handles missing values gracefully with the special constant and reserved keyword `NA`.

summarize

Find the sum of two columns. Ignore NA values in the x column.

```
CrossFit %>% summarise(x.total = sum(x, na.rm = TRUE), n.total = sum(n)) %>%
  mutate(p.total = x.total / n.total) %>% kable()
```

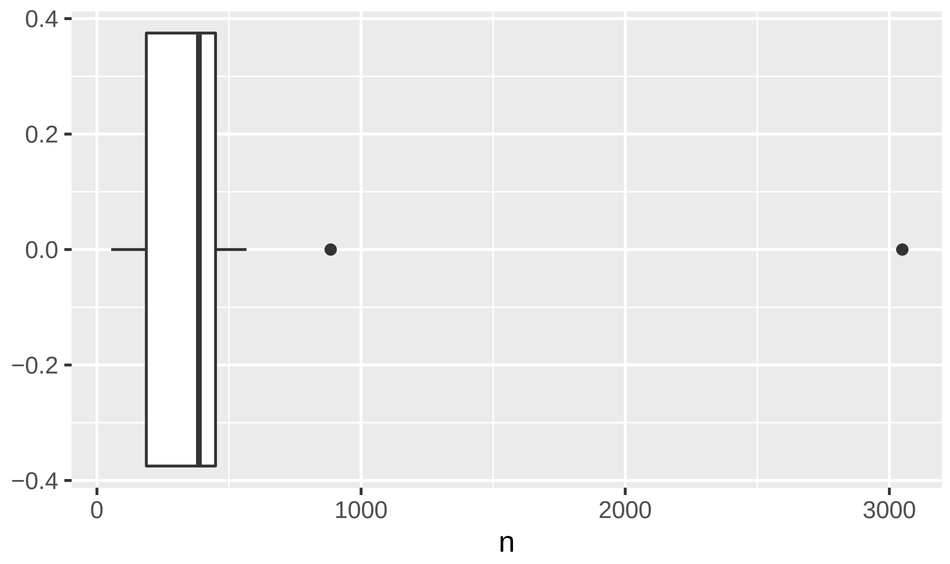
x.total	n.total	p.total
2453	8361	0.293386

ggplot2

ggplot2 is a powerful tool for creating beautiful plots. It is not trivial to learn, but you get away with searching the Internet for what you need. 'ggplot2 can accept input from the %>% operator, but you use the + operator to compose ggplot2 functions. ggplot2 *can* produce pie charts, but apparently pie charts have fallen out of favor. Here are some simple examples:

Box Plot

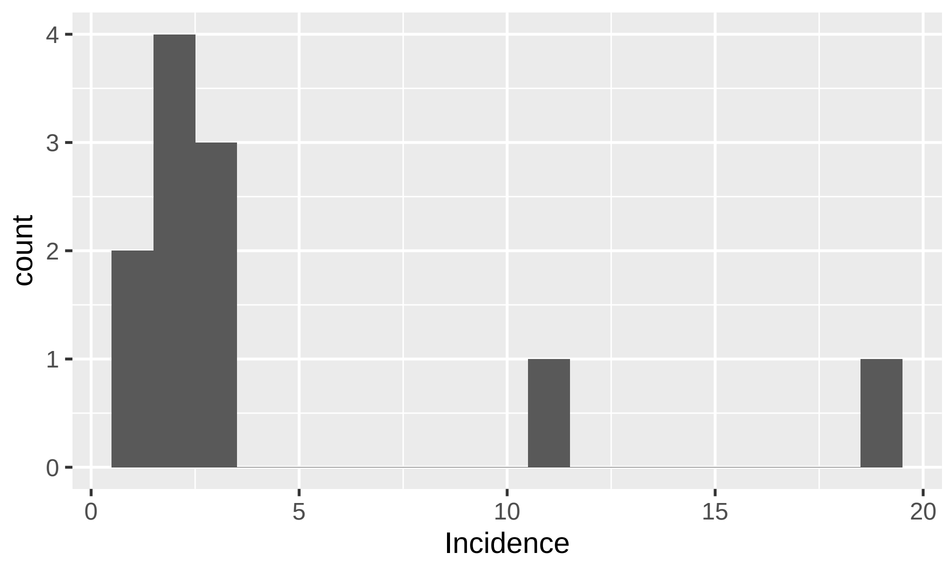
```
CrossFit %>% ggplot(mapping = aes(x = n)) + geom_boxplot()
```



Histogram

```
CrossFit %>% ggplot(aes(Incidence)) + geom_histogram(binwidth = 1)
```

```
## Warning: Removed 6 rows containing non-finite values (stat_bin).
```

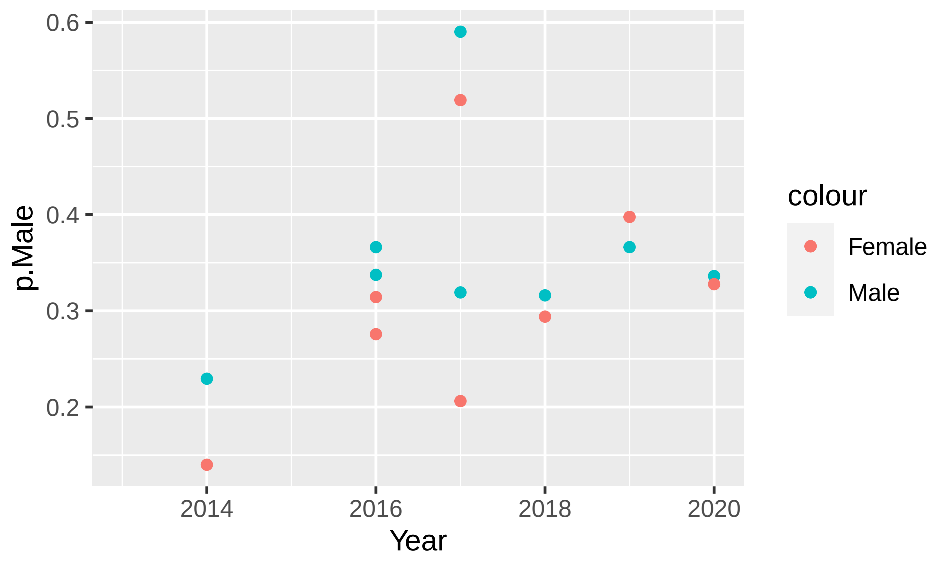


Points

```
CrossFit %>% ggplot(aes(x = Year)) + geom_point(aes(y = p.Male, color="Male")) +
  geom_point(aes(y = p.Female, color="Female"))
```

```
## Warning: Removed 9 rows containing missing values (geom_point).

## Warning: Removed 9 rows containing missing values (geom_point).
```

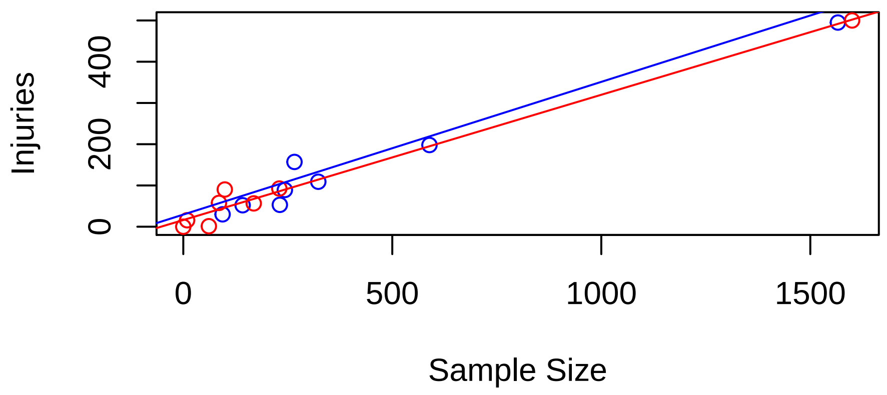


Basic Plots

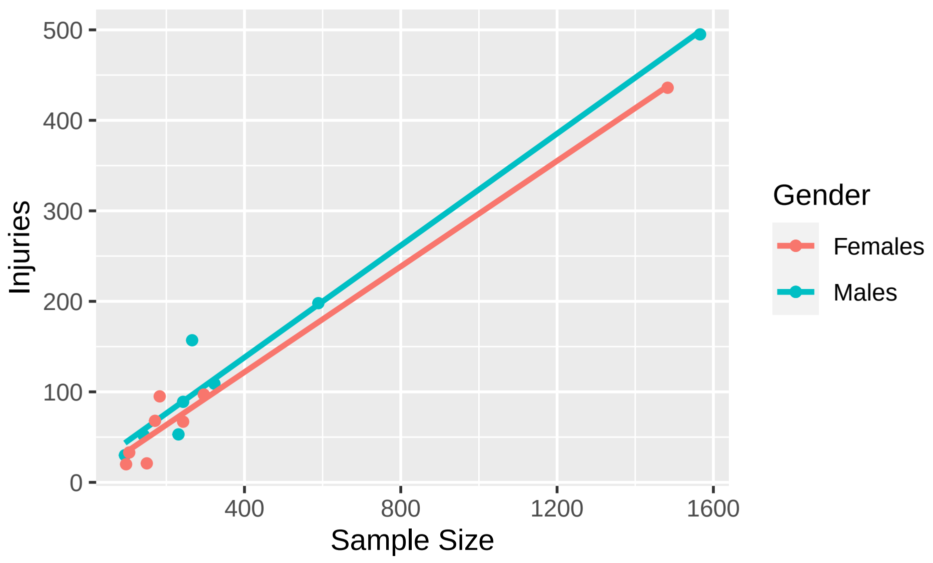
Vanilla R also has basic plotting utilities, such as `pie`, `hist`, `plot`, and `boxplot`. These functions have a little bit less functionality than `ggplot2` and (at least by default) and less pretty, but the commands are simpler and easier to figure out. The basic functions are actually more difficult to use when you want something that isn't trivial. For example, compare a

Mathematica's `Show` command is a little easier to use. In R, it is not completely obvious how to show multiple series on a single plot. Here is one way:

```
# Base R
Injuries <- CrossFit %>% select(starts_with("x"), starts_with("n")) %>% drop_na()
plot(Injuries$n.Male, Injuries$x.Male, xlab = "Sample Size", ylab = "Injuries",
     col = "blue", xlim = c(0,1600), ylim = c(0,500))
par(new = TRUE)
plot(Injuries$n.Female, Injuries$x.Female, xlab = "", ylab = "", col = "red",
     axes = FALSE)
abline(lm(Injuries$x.Male ~ Injuries$n.Male), col = "blue")
abline(lm(Injuries$x.Female ~ Injuries$n.Female), col = "red")
```



```
# ggplot2, arguably simpler for non-trivial plots.
Injuries %>% ggplot() + geom_point(aes(n.Male, x.Male, color = "Males")) +
  geom_smooth(method = lm, formula = y ~ x, aes(n.Male, x.Male, color = "Males"),
              se = FALSE) +
  geom_point(aes(n.Female, x.Female, color = "Females")) +
  geom_smooth(method = lm, formula = y ~ x, aes(n.Female, x.Female, color = "Females"),
              se = FALSE) +
  xlab("Sample Size") +
  ylab("Injuries") +
  labs(color = "Gender")
```



R internals

If ever you need to negate any “restored my faith in humanity” clickbait then have a look in R’s source code. Here is `pnorm.c`. Now I appreciate why it is an advantage for much of Julia to be itself implemented in Julia.

List Comprehension

I didn’t initially appreciate list comprehension in Python when I first encountered it. Now I love it. Mathematica has a similar concept in its Array function and Julia provides the same syntax as Python. There is a `comprehenr` library that gives R the same functionality.

```
fibonacci <- function(n) {
  if (n == 0 | n == 1)
    return(1)
  else
    return(fibonacci(n-1) + fibonacci(n-2))
}
to_vec(for(i in 0:20) fibonacci(i))
```

```
##  [1]     1     1     2     3     5     8    13    21    34    55    89   144
## [13]   233   377   610   987  1597  2584  4181  6765 10946
```

Basic Statistics

Ok, so I have talked about R and compared it to a few other programming languages that I am comfortable with. The remainder of this document is basically my notes for my future self to refer to once I have forgotten everything.

Basic Stats

```
WorldSeries <- data.frame(Year = 1903:2019,
                          Length = c(8, NA, 5, 6, 5, 5, 7, 5, 6, 8, 5, 4, 5, 5, 6, 6, 8,
                                     7, 8, 5, 6, 7, 7, 7, 4, 4, 5, 6, 7, 4, 5, 7, 6, 6, 5,
                                        4, 4, 7, 5, 5, 5, 6, 7, 7, 7, 6, 5, 4, 6, 7, 6, 4, 7,
                                        7, 7, 7, 6, 7, 5, 7, 4, 7, 7, 4, 7, 7, 5, 5, 7, 7, 7,
                                        5, 7, 4, 6, 6, 7, 6, 6, 7, 5, 5, 7, 7, 7, 5, 4, 4, 7,
                                        6, 6, NA, 6, 6, 7, 4, 4, 5, 7, 7, 6, 4, 4, 5, 4, 5,
                                        6, 5, 7, 4, 6, 7, 5, 7, 7, 5, 7))
# see also mean(), median(), and range()
summary(WorldSeries)
```

```
##       Year          Length     
##  Min.   :1903   Min.   :4.000  
##  1st Qu.:1932   1st Qu.:5.000  
##  Median :1961   Median :6.000  
##  Mean   :1961   Mean   :5.843  
##  3rd Qu.:1990   3rd Qu.:7.000  
##  Max.   :2019   Max.   :8.000  
##                 NA's   :2
```

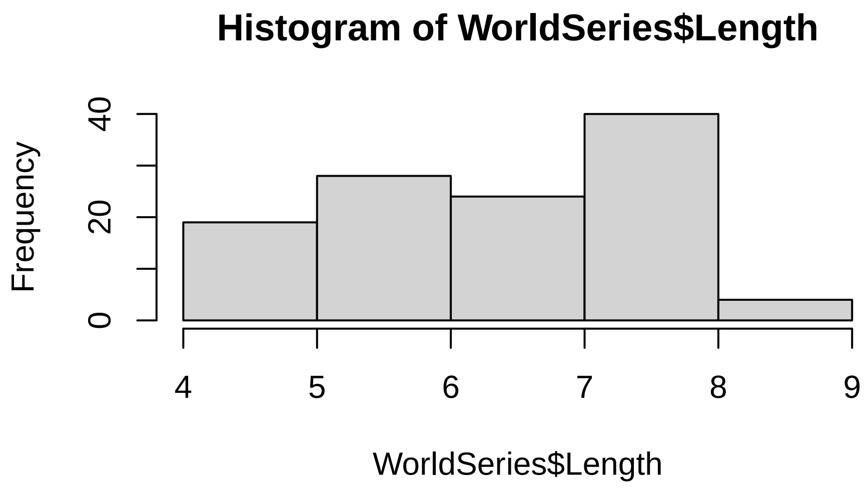
```
# variance and sample standard deviation
var(WorldSeries$Length, na.rm = TRUE)
```

```
## [1] 1.378795
```

```
sd(WorldSeries$Length, na.rm = TRUE)
```

```
## [1] 1.174221
```

```
# the hist() function returns an object with many useful fields.
h <- hist(WorldSeries$Length, breaks = 4:9, right = FALSE)
```



```
data.frame(Length = (h$breaks)[-length(h$breaks)],
           Frequency = h$counts,
           Relative = h$density,
           Cumulative = cumsum(h$density)) %>% kable()
```

Length	Frequency	Relative	Cumulative
4	19	0.1652174	0.1652174
5	28	0.2434783	0.4086957

Length	Frequency	Relative	Cumulative
6	24	0.2086957	0.6173913
7	40	0.3478261	0.9652174
8	4	0.0347826	1.0000000

Discrete Probability

The Binomal Formula for the probability of r successes in n trials is

$$P(x = r) = \binom{n}{r} p^r q^{n-r}.$$

```
# combination
for (i in 0:8) {
  print(choose(i, 0:i))
}
```

```
## [1] 1
## [1] 1 1
## [1] 1 2 1
## [1] 1 3 3 1
## [1] 1 4 6 4 1
## [1]  1  5 10 10  5  1
## [1]  1  6 15 20 15  6  1
## [1]  1  7 21 35 35 21  7  1
## [1]  1  8 28 56 70 56 28  8  1
```

```
# factorial
factorial(0:8)
```

```
## [1]     1     1     2     6    24   120   720  5040 40320
```

```
# probability of getting EXACTLY 0-10 heads of 10 fair die rolls.
dbinom(x = 0:10, size = 10, prob = 1/6)
```

```
##  [1] 1.615056e-01 3.230112e-01 2.907100e-01 1.550454e-01 5.426588e-02
##  [6] 1.302381e-02 2.170635e-03 2.480726e-04 1.860544e-05 8.269086e-07
## [11] 1.653817e-08
```

```
sum(dbinom(x = 0:10, size = 10, prob = 1/6))
```

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

```
# cumulative probability of getting AT LEAST 0-10 heads.
cumsum(dbinom(x = 0:10, size = 10, prob = 1/6))
```

```
##  [1] 0.1615056 0.4845167 0.7752268 0.9302722 0.9845380 0.9975618 0.9997325
##  [8] 0.9999806 0.9999992 1.0000000 1.0000000
```

```
pbinom(q = 0:10, size = 10, prob = 1/6)
```

```
##  [1] 0.1615056 0.4845167 0.7752268 0.9302722 0.9845380 0.9975618 0.9997325
##  [8] 0.9999806 0.9999992 1.0000000 1.0000000
```

```
# probability of getting MORE THAN 0-10 heads.
pbinom(q = 0:10, size = 10, prob = 1/6, lower.tail = FALSE)
```

```
##  [1] 8.384944e-01 5.154833e-01 2.247732e-01 6.972784e-02 1.546197e-02
##  [6] 2.438156e-03 2.675215e-04 1.944889e-05 8.434468e-07 1.653817e-08
## [11] 0.000000e+00
```

```r
1 - pbinom(q = 0:10, size = 10, prob = 1/6)
```

```
##  [1] 8.384944e-01 5.154833e-01 2.247732e-01 6.972784e-02 1.546197e-02
##  [6] 2.438156e-03 2.675215e-04 1.944889e-05 8.434468e-07 1.653817e-08
## [11] 0.000000e+00
```

```r
# inverse of cumulative distribution function:
# how many heads are in p% of 10 fair rolls?
qbinom(p = seq(0, 1, .1), size = 10, prob = 1/6)
```

```
##  [1]  0  0  1  1  1  2  2  2  3  3 10
```

In a binomial distribution $\mu = np$ and $\sigma^2 = nqp$ where $q = 1 - p$.

Continuous Probability

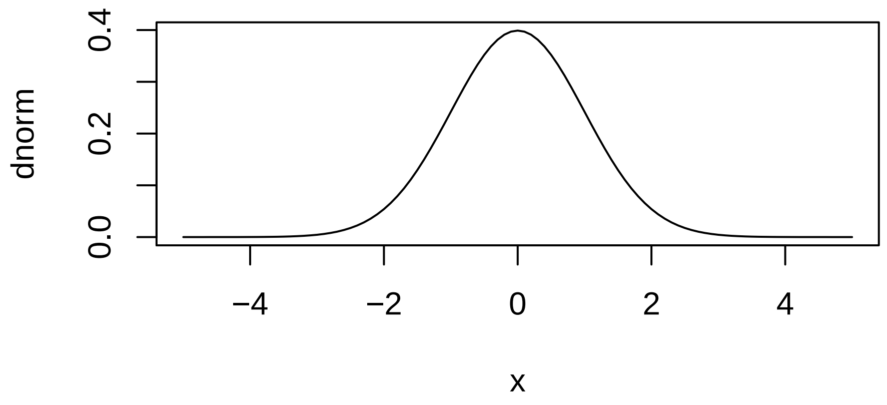
The normal distribution's probability density function is

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

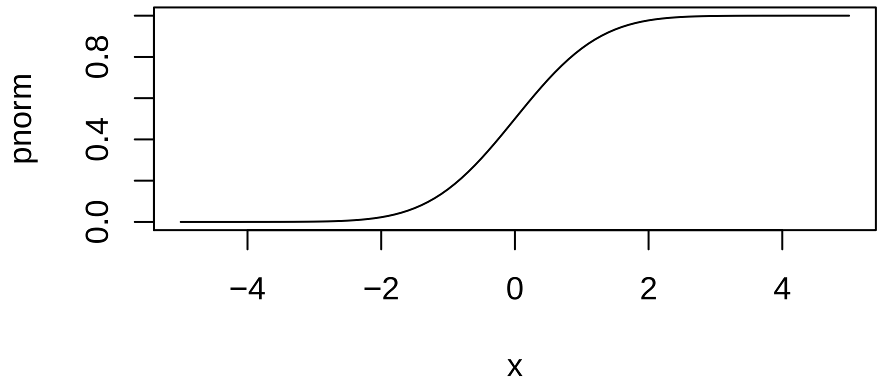
A z-score is computed from

$$z = \frac{x - \mu}{\sigma}.$$

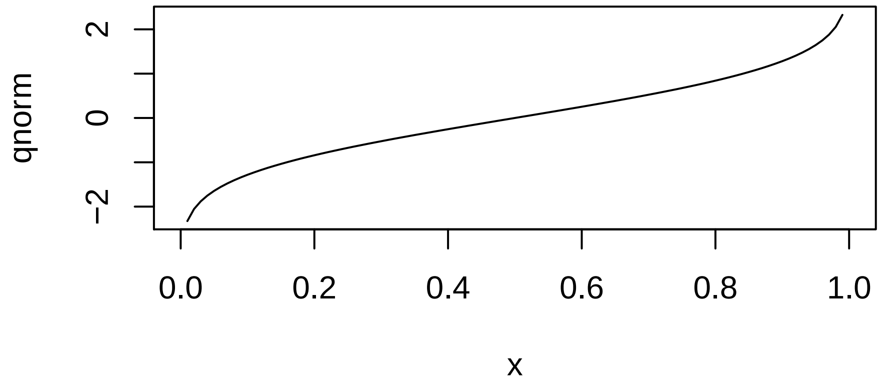
```r
# probability density function for a normal distribution of default mean = 0 and sd = 1.
plot(dnorm, xlim = c(-5,5))
```



```r
# cumulative probability
plot(pnorm, xlim = c(-5,5))
```



```r
# inverse of cumulative probability
plot(qnorm)
```



```r
# Empirical Rule 68% of data lies within +/-1sd, 95% +/-2sd, 99.7% +/-3sd.
1 - 2 * pnorm(-1:-3)
```

```
## [1] 0.6826895 0.9544997 0.9973002
```

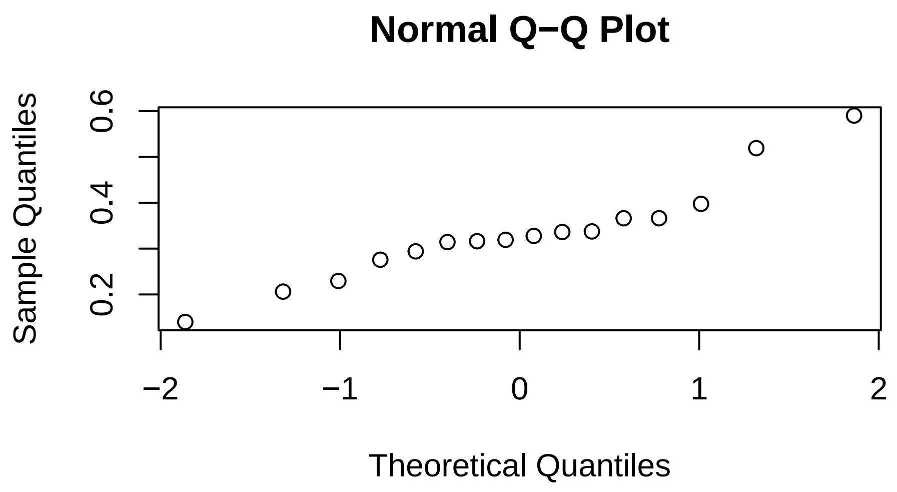
```r
# if the average of some normal distribution is 100 and standard deviation is 20,
# what is the probability of a quantity between 60 and 90?
pnorm(q = 90, mean = 100, sd = 20) - pnorm(q = 60, mean = 100, sd = 20)
```

```
## [1] 0.2857874
```

```r
integrate(function(x) dnorm(x, mean = 100, sd = 20), lower = 60, upper = 90)
```

```
## 0.2857874 with absolute error < 3.2e-15
```

```r
# normal probability plot. This was not really emphasized in my textbook. It is a way to
# assess if your data forms a normal distribution.
qqnorm(c(CrossFit$p.Male, CrossFit$p.Female))
```



Central Limit Theorem: suppose a random variable is from any distribution. If a sample of size n is taken, then the sample mean, $\bar{x}$, becomes normally distributed as n increases.

The rest

The remainder of this paper is an extremely quick review of some important statistical functions in R. I have run out of time and cannot make this review as comprehensive as I would like. Forgive me, future self!

I have run out of time to write these notes. So far I have only covered the first half of the class. The remainder of the class covered the 1-prop test, 1 sample test for the mean (t-Test), 1-prop interval, t-interval, 2-prop test, 2-sample paired t-Test, 2-sample t-Test, 2-sample t-interval, regression, r^2, χ^2 test, and ANOVA test.

Here are the short descriptions.

t-Test

The t test allows you to reason about small samples. R contains the functions `dt`, `pt`, and `qt` that largely work the same way as `dnorm`, `pnorm`, and `qnorm`. The major difference is the *degrees of freedom*. Usually, $df = n - 1$. You also have the wonderful *t.test* (which can accept a `paired` parameter) which does everything for you.

`t.test` can help you answer whether $\mu_1 = \mu_2$ and estimate the difference in means,

$$(\bar{x}_1 - \bar{x}_2) - E < \mu_1 - \mu_2 < (\bar{x}_1 - \bar{x}_2) + E.$$

```r
t.test(CrossFit$p.Male, CrossFit$p.Female)
```

```
##
##  Welch Two Sample t-test
##
## data:  CrossFit$p.Male and CrossFit$p.Female
## t = 0.88263, df = 13.839, p-value = 0.3925
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.0691794  0.1657546
## sample estimates:
## mean of x mean of y
## 0.3576225 0.3093349
```

```
# we get a completely different result when we pair the male/female statistics by study.
t.test(CrossFit$p.Male, CrossFit$p.Female, paired = TRUE)
```

```
##
##  Paired t-test
##
## data:  CrossFit$p.Male and CrossFit$p.Female
## t = 2.9296, df = 7, p-value = 0.02204
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.009312098 0.087263152
## sample estimates:
## mean of the differences
##              0.04828762
```

Proportion test

`prop.test` lets you compare proportions of "successes" and determine if they are the same.

```
# unsurprisingly, some studies are have MUCH higher proportions than others.
prop.test(CrossFit$x.Male, CrossFit$n.Male)
```

```
##
##  8-sample test for equality of proportions without continuity
##  correction
##
## data:  CrossFit$x.Male out of CrossFit$n.Male
## X-squared = 91.782, df = 7, p-value < 2.2e-16
## alternative hypothesis: two.sided
## sample estimates:
##    prop 3    prop 5    prop 6    prop 8    prop 9   prop 11   prop 13   prop 15
## 0.2294372 0.3661972 0.3374613 0.5902256 0.3191489 0.3160920 0.3662551 0.3361630
```

You can also use `prop.test` to find confidence intervals.

```
# confidence interval from third study
prop.test(CrossFit$x[3], CrossFit$n[3], conf.level = 0.90)
```

```
##
##  1-sample proportions test with continuity correction
##
## data:  CrossFit$x[3] out of CrossFit$n[3], null probability 0.5
## X-squared = 143.07, df = 1, p-value < 2.2e-16
## alternative hypothesis: true p is not equal to 0.5
## 90 percent confidence interval:
##  0.1621469 0.2308752
## sample estimates:
##         p
## 0.1943005
```

This means that, from the data from the third study, there is a 90% probability that the parameter proportion of athletes injured in CrossFit is on the calculated interval.

Regression

R makes linear regressions really easy.

```r
model <- lm(CrossFit$x.Male ~ CrossFit$n.Male)
summary(model)
```

```
##
## Call:
## lm(formula = CrossFit$x.Male ~ CrossFit$n.Male)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -32.831  -8.115  -4.347  -0.023  60.352
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     14.43716   14.11245   1.023    0.346
## CrossFit$n.Male  0.30906    0.02261  13.667 9.53e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 28.82 on 6 degrees of freedom
##   (9 observations deleted due to missingness)
## Multiple R-squared:  0.9689, Adjusted R-squared:  0.9637
## F-statistic: 186.8 on 1 and 6 DF,  p-value: 9.531e-06
```

χ^2 Test

The χ^2 test can tell if two variables are dependent using the equation

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

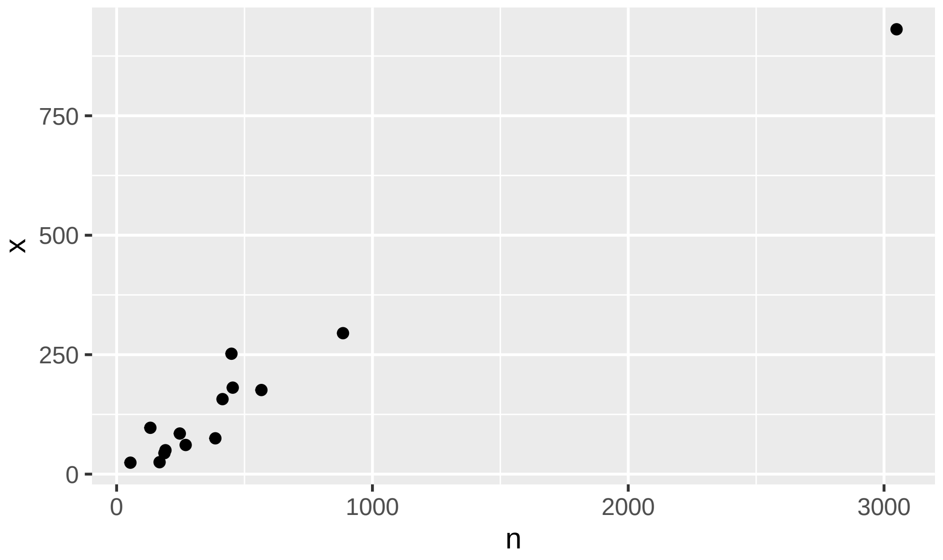
where O is the observed frequency and E is the expected frequency.

```r
# Is injury prevalence equal across all studies? Probably not.
chisq.test(x = CrossFit$x, y = CrossFit$n)
```

```
## Warning in chisq.test(x = CrossFit$x, y = CrossFit$n): Chi-squared approximation
## may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  CrossFit$x and CrossFit$n
## X-squared = 182, df = 169, p-value = 0.234
```

```r
CrossFit %>% select(x, n) %>% drop_na() %>% ggplot(aes(n, x)) + geom_point()
```



ANOVA

What is the probability that the mean engine size is the same for all number of cylinders?

```
aov(formula = disp ~ cyl, data = mtcars)
```

```
## Call:
##    aov(formula = disp ~ cyl, data = mtcars)
##
## Terms:
##                       cyl Residuals
## Sum of Squares  387454.1   88730.7
## Deg. of Freedom        1        30
##
## Residual standard error: 54.38465
## Estimated effects may be unbalanced
```

```
summary(aov(formula = disp ~ cyl, data = mtcars))
```

```
##             Df Sum Sq Mean Sq F value  Pr(>F)
## cyl          1 387454  387454     131 1.8e-12 ***
## Residuals   30  88731    2958
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Not very likely.

Version

This document was created using:

```
version
```

```
##                  _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          4
## minor          0.0
## year           2020
## month          04
## day            24
## svn rev        78286
## language       R
## version.string R version 4.0.0 (2020-04-24)
## nickname       Arbor Day
```