R Companion

STAT 315

Getting Started

R is the base version of a free, open-source statistical computing program. R Studio is a nice graphic user interface (GUI) that works in conjunction with R. We will be using R Studio this semester as it is easier to learn how to use and has many nice shortcuts that aren't available in the standard R download.

Note: R and R Studio can not be installed on Chromebooks or other tablets/mini-computers. If you only have a tablet or a mini-computer, you can use R Studio Cloud

To begin using R Studio, you will need to install "R" first and then install "R Studio" on your computer.

Step 1: Download R
(a): Visit `https://www.r-project.org/`
(b): Click **CRAN** under **Download**
(c): Select any of the mirrors
(d): Click the appropriate link for your type of system (Mac, Windows, Linux)
(e): Download R on this next page.
(For Windows, this will say **install R for the first time**. For Mac, this will be under **Latest release** and will be something like **R-4.1.0.pkg** – the numbers may differ depending on the most recent version)
(f): Install R on your computer

Step 2: Download R Studio
(a): Visit `https://www.rstudio.com/products/rstudio/download/#download`
(b): Click to download
(c): Install R Studio on your computer

Step 3: Verify R Studio is working

(a): Open R Studio

(b): Let's enter a small dataset and calculate the average to make sure everything is working correctly.

(c): In the console, type in the following dataset of five homework scores:

```r
x = c(90,75,100,85,90)
```

(d): In the console, calculate the average of the five homework scores:

```r
mean(x)

## [1] 88
```

(e) Did you find the average of the five homework scores was 88? If so, you should be set up correctly!

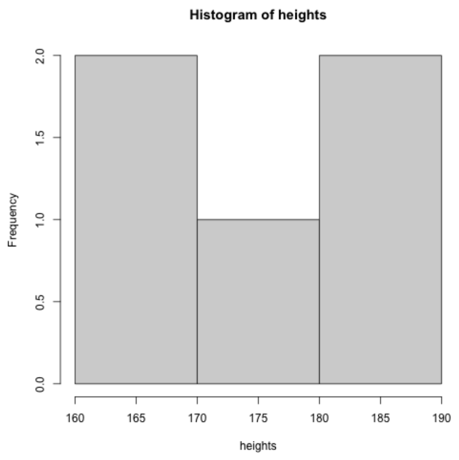
Chapter 1

Example 1: Let's manually enter the heights data set (183, 170, 160, 175, 187) from chapter 1 and the following R commands will do the calculations we completed by hand.
Note: Lines of code that begin with # are comments and are not executed by the computer. Lines of code that begin with # denote the output.

```
# manually enter your data by using the "c" command
heights = c(183,170,160,175,187)

# calculate sample statistics on data set
mean(heights)

## [1] 175

median(heights)

## [1] 175

var(heights)

## [1] 114.5

sd(heights)

## [1] 10.70047
```

We can also create a histogram in R. Note that we can specify the breaks for the bins. In this case, the breaks are defined at 160, 170, 180, 190.

```
hist(heights, breaks = c(160, 170, 180, 190))
```



Example 2: Let's create a randomized dataset of eye colors and analyze the data.

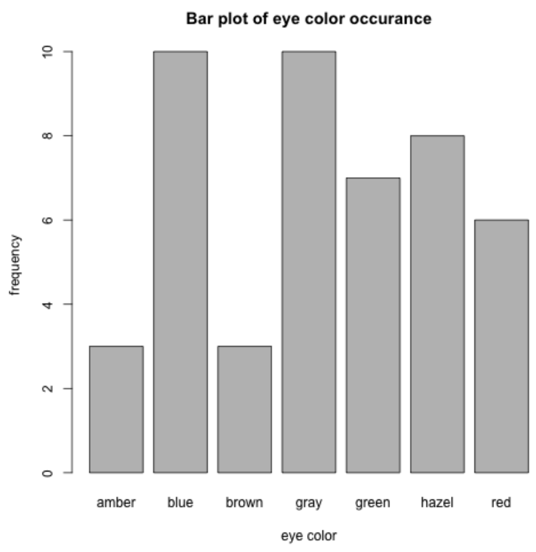
Note: When we generate random data in this class, we will use the **set.seed** function, so that we all generate the same random data. If we don't use the **set.seed** command before generating random data, we will all get different random data.

Here, we generate 47 random eye colors and take a look at the first few observations.

```
set.seed(2020)
eye.colors <- sample(c("amber", "blue", "brown", "gray", "green", "hazel",
    "red"), 47, replace = T)

# the 'head' function will give the first few data values or the
# 'header' of the dataset
head(eye.colors)

## [1] "gray"  "gray"  "red"   "hazel" "amber" "amber"
```

We can create a frequency table of these values and create a barplot.

```
table(eye.colors)

## eye.colors
## amber  blue brown  gray green hazel   red
##     3    10     3    10     7     8     6

barplot(table(eye.colors), xlab = "eye color",
        ylab = "frequency", main = "Bar plot of eye color occurance")
```



We can also calculate proportion and percentage tables.

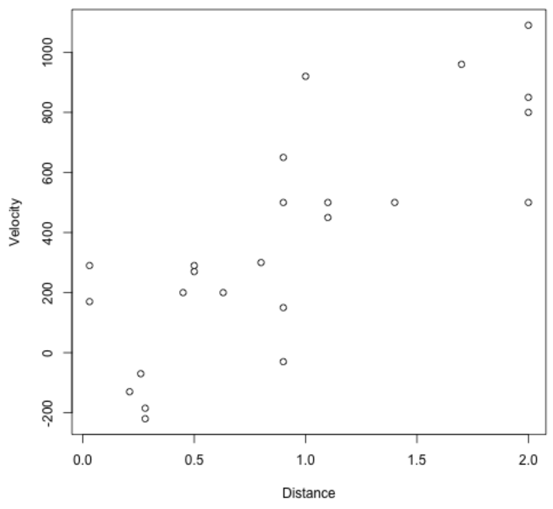
```
table(eye.colors) # shows how many of each value exist

## eye.colors
## amber  blue brown  gray green hazel   red
##     3    10     3    10     7     8     6

length(eye.colors) # tells how many data points there are

## [1] 47

table(eye.colors)/length(eye.colors) # proportion

## eye.colors
##      amber       blue      brown       gray      green      hazel
## 0.06382979 0.21276596 0.06382979 0.21276596 0.14893617 0.17021277
##        red
## 0.12765957

table(eye.colors)/length(eye.colors)*100 # percent

## eye.colors
##     amber      blue     brown      gray     green     hazel       red
##  6.382979 21.276596  6.382979 21.276596 14.893617 17.021277 12.765957
```

Example 3: We can easily load datasets into R Studio. Let's load **hubble.csv** which are data on distances and velocities of 24 galaxies containing Cepheid stars, from the Hubble space telescope key project to measure the Hubble constant.

First, click **File** in the menu bar, then **Import Dataset**, then **From Text (base)**, and finally find and select **hubble.csv**.

Let's plot distance versus velocity. Notice that we can identify a possible relationship between the two variables in the plot.

```
plot(Velocity~Distance,data=hubble)
```



If you want to examine one of the variables defined under the **hubble** dataset, you can use the **$** to refer to the variable of interest. For example, we can calculate the mean of Distance and variance of Velocity as follows.

```
mean(hubble$Distance)

## [1] 0.91125

var(hubble$Velocity)

## [1] 137830
```

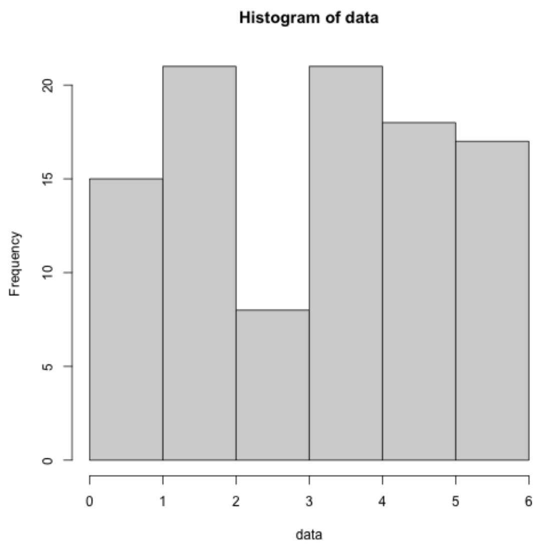
Chapter 2

Example 1: Let's simulate a fair six-sided die roll. Recall that we need to use the **set.seed** function so that we all generate the same random data. For consistency, I always set the seed to 2020, i.e., set.seed(2020). The sample function will generate 100 random die rolls with each possible outcome (1,2,3,4,5,6) being equally likely.

```
set.seed(2020)
data = sample(x = 1:6,size = 100,replace = T); data

##   [1] 4 4 6 1 1 4 2 6 1 5 2 2 6 5 2 3 2 5 4 2 6 6 4 6 4 2 4 5 4 4 3 6
##  [33] 2 2 6 3 5 4 5 5 2 5 1 6 3 5 1 5 3 1 5 3 2 6 2 1 3 2 1 6 5 5 2 5
##  [65] 6 4 3 6 4 4 2 2 1 6 1 4 6 2 2 2 1 4 1 1 4 2 2 5 5 4 4 4 6 5 4 1
##  [97] 6 5 1 4
```

Let's make a histogram of the simulated data. Recall that each outcome (1,2,3,4,5,6) is equally likely in theory but in our simulation, some numbers may come up more than others. To make the histogram look better, I define the boundary breaks to be 0:6 (i.e., 0,1,2,3,4,5,6).

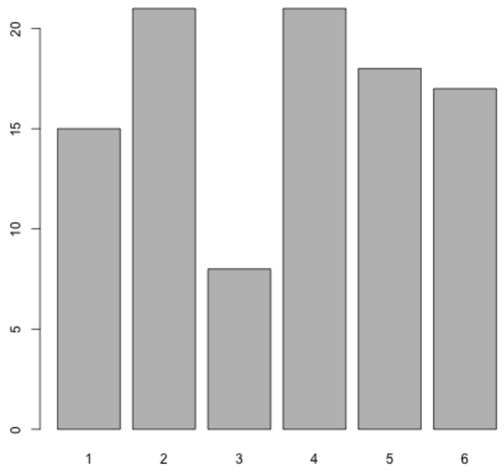
```
hist(data,breaks=0:6)
```



With only 100 random die rolls, the distribution isn't as flat, uniform, and symmetric as we would expect. The number 3 came up less than we would anticipate. This is OK.

We can also generate a frequency table and use this to create a barplot. This looks a little nicer than the standard histogram and doesn't require use to use the **breaks** argument.

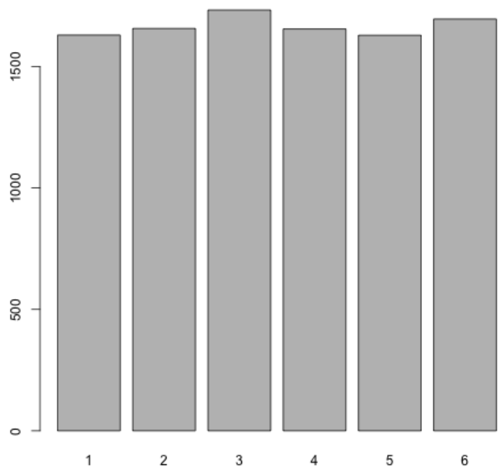
```
table(data)

## data
##  1  2  3  4  5  6
## 15 21  8 21 18 17

barplot(table(data))
```



If we want a better representation of the true distribution of the fair six-sided die rolls, we could simulate 10000 die rolls instead of 100.

```
set.seed(2020)
data = sample(x = 1:6,size = 10000,replace = T)
table(data)

## data
##    1    2    3    4    5    6
## 1630 1657 1733 1655 1629 1696
```

```
barplot(table(data))
```



Simulation can be helpful for us to verify hand calculations and to estimate quantities that are hard to calculate by hand. Our simulation results won't exactly match our hand calculations, but if the sample size is large enough, they should be close.

For instance, we know that P(Roll=1) = 1/6 for the population of all fair six-sided die rolls. If you refer to the table at the bottom of the previous page, you see that 1 came up 1630 times out of 10000 die rolls. In the simulation, P(1) = 1630/10000 = 0.1630 which is pretty close the the theoretical value of P(1) = 1/6 = 0.1667.

Here is an example of how to calculate the probability of a die roll less than 3 in our simulation.

```
# Calculate P(Roll < 3)
# From theory, we know that P(Roll < 3) = P(1) + P(2) = 1/6 + 1/6 = 1/3

# count up the number of rolls less than 3
num.under3 = sum( data < 3 )

# divide the number of rolls less than 3 by the number of rolls
prob.under3 = num.under3 / 10000
prob.under3

## [1] 0.3287
```

From theory, we expect $P(\text{Roll} < 3) = 1/3 = 0.3333$. In our simulation, we found $P(\text{Roll} < 3) = 0.3287$.

Example 2: Let's simulate rolling two fair six-sided dice 10,000 times and calculate some probabilities. **d1** will represent the first die roll and **d2** will represent the second die roll.

```
set.seed(2020)
d1 = sample(x = 1:6,size = 10000,replace = T)
d2 = sample(x = 1:6,size = 10000,replace = T)
```

From our simulated data, let's calculate P(D1 = 1 and D2 = 2), that is, the probability that first die roll is 1 and the second die roll is 2. From theory, this probability should be $1/36 = 0.0278$. (We use the **&** for **AND** in R.)

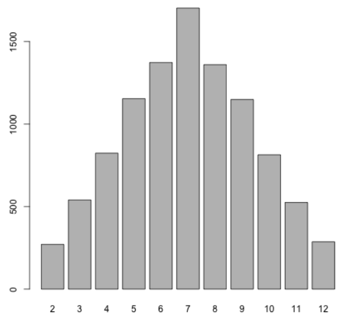
```
num.events = sum(d1 == 1 & d2 == 2)
prob = num.events / 10000; prob

## [1] 0.0256
```

Next, let's calculate P(D1 < 2 or D2 > 2), that is, the probability that the first die roll is less than 2 or the second die roll is greater than 2. From theory, this probability should be 0.7222. We use the | for the **OR** in R.)

```
num.events = sum(d1 < 2 | d2 > 2)
prob = num.events / 10000; prob

## [1] 0.7183
```

Finally, let's look at the frequency table and distribution of the sum of the two die rolls.

```
table(d1+d2)

##
##    2    3    4    5    6    7    8    9   10   11   12
##  271  540  824 1154 1373 1703 1360 1149  814  525  287

barplot(table(d1+d2))
```



Chapter 3

We can calculate probabilities for common discrete random variables easily in R. There are two parts to finding probabilities using the PMF or CDF.
R uses four prefixes to reference difference elements of a random variable. These are:

p for "probability", the cumulative distribution function (CDF)
q for "quantile", the inverse CDF
d for "density", the probability mass function (PMF)
r for "random", a random variable having the specified distribution

In addition, we have suffixes for common random variables: **binom** (binomial), **pois** (Poisson), and **geom** (Geometric).

For instance, suppose that $X \sim Binomial(n = 10, p = 0.5)$, that is, X is a binomial random variable that counts the number of heads in 10 fair coin flips. We can calculate P(X=5), the probability of exactly 5 heads in 10 fair coin flips, as follows.

```
dbinom(x=5,size=10,prob=0.5)

## [1] 0.2460938
```

If we want the probability of at most five heads in 10 coin flips, $P(X \leq 5)$, we can use **pbinom**.

```
pbinom(q=5,size=10,prob=0.5)

## [1] 0.6230469
```

Now, suppose that $Y \sim Poisson(\lambda = 2)$ and we want to know $P(Y > 3)$. First, we note that $P(Y > 3) = 1 - P(Y \leq 3)$.

```
1-ppois(q=3,lambda=2)

## [1] 0.1428765
```

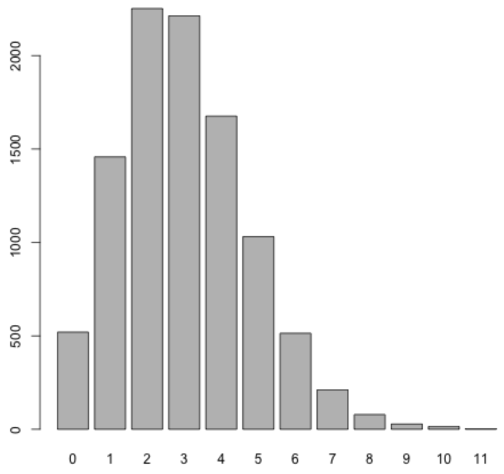
If we use the prefix **r**, we can generate simulated data according to a particular random variable. Let's generate 10,000 observations from a Poisson distribution with $\lambda = 3$, look at the first few values, and then calculate some summary statistics. This will require us to use the **rpois** function to generate random Poisson data.

```
rand.data = rpois(n=10000,lambda=3)
head(rand.data) # look at the first few values
```

```
## [1] 3 6 2 4 1 4

mean(rand.data) # calculate the mean

## [1] 3.0088

var(rand.data) # calculate the variance

## [1] 3.047827
```

From theory, we know that if $Y \sim Poisson(\lambda = 3)$, then $E[Y] = Var(Y) = \lambda = 3$. Our simulated results are pretty close to the theoretical values.

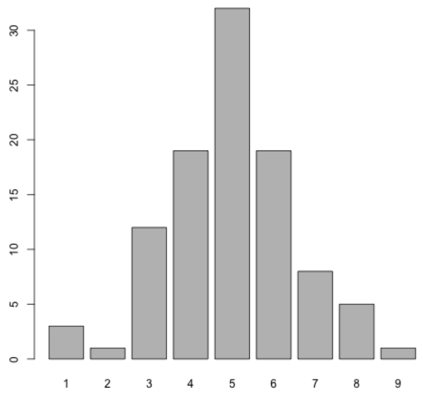
```
table(rand.data)

## rand.data
##    0    1    2    3    4    5    6    7    8    9   10   11
##  520 1458 2252 2213 1676 1031  514  211   79   28   15    3

barplot(table(rand.data))
```



How would you describe the distribution above? It seems to be asymmetric, skewed right (tail on the right), and unimodal.

Example: Let's simulate some data and calculate the probabilities in the sample. Suppose we flip 10 fair coins and we count the number of heads. Let X be the number of heads. We can model X as a binomial random variable with parameters n=10 (number of coin flips) and p=0.5 (fair coin). Let's simulate running this experiment 100 times.

```
set.seed(2020)
coins = rbinom(n=100,size=10,prob=0.5)
table(coins)

## coins
##  1  2  3  4  5  6  7  8  9
##  3  1 12 19 32 19  8  5  1

barplot(table(coins))
```



In our simulation, what was the probability that 5 heads occurred out of 10 flips? In the above table, we can see we got 5 heads on 32 of the 100 simulations, so $P(5) = 32/100 = 0.32$.

We could also calculate as follows:

```
p5 = sum(coins == 5)/100; p5

## [1] 0.32
```

What is the probability that we got more than 7 heads?

```
p = sum(coins > 7)/100; p

## [1] 0.06
```

Chapter 4

Similar to last chapter, we can calculate probabilities for common continuous random variables in R and generate random data according to continuous distributions.

R uses four prefixes to reference difference elements of a random variable. These are:

p for "probability", the cumulative distribution function (CDF)
q for "quantile", the inverse CDF
d for "density", the probability mass function (PMF)
r for "random", a random variable having the specified distribution

Suffixes for continuous random variables include: **norm** (normal/Gaussian), **exp** (exponential), and **t** (t).

Suppose $X \sim \mathcal{N}(\mu = 100, \sigma = 15)$, the distribution often used to model IQ scores. Suppose we want to know the probability of a randomly chosen IQ score below 110.

```
pnorm(q=110,mean=100,sd=15)

## [1] 0.7475075
```

In other words, about 75% of IQ scores are less than 110.

Let's calculate the probability that a randomly chosen score is above 130. Note that $P(X > 130) = 1 - P(X \leq 130)$.

```
1-pnorm(q=130,mean=100,sd=15)

## [1] 0.02275013
```

In other words, about 2.3% of IQ scores are greater than 130.

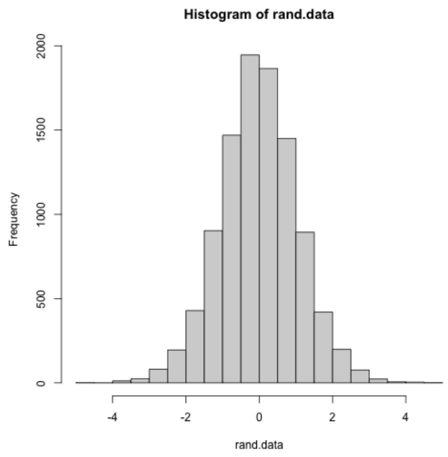
We can also do inverse normal calculations using the prefix **q** which stands for quantile. Suppose we want to know the 90th percentile of IQ scores. We can calculate this using the **qnorm** function.

```
qnorm(p=0.9,mean=100,sd=15)

## [1] 119.2233
```

This result shows that the 90th percentile of IQ scores corresponds to an approximate IQ of 119.

In the second half of the course, we will be using the **t-distribution** frequently when dealing with statistical applications. Let's generate 10,000 random values for a t-distribution with 20 degrees of freedom, i.e., $Z \sim t(df = 20)$.

```
set.seed(2020)
rand.data = rt(n=10000,df=20)
hist(rand.data)
```



Based on the plot above, this distribution is approximately symmetric, bell-shaped, and unimodal.

Let's calculate some summary statistics for this data.

```
mean(rand.data)

## [1] -0.009077007

median(rand.data)

## [1] -0.01514699
```

The mean and median are both approximately equal to zero and since these values are nearly equal, we can confirm that the distribution is approximately symmetric.
Finally, let's calculate the probability in our simulated data so that $P(Z > 1)$.

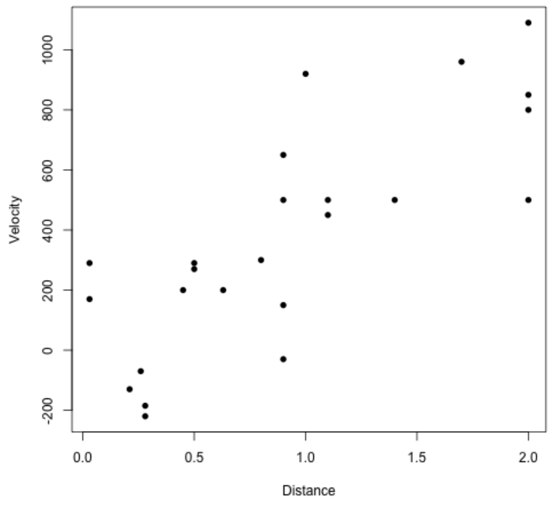
```
num.events = sum(rand.data > 1)
prob = num.events / 10000; prob

## [1] 0.1623
```

Approximately 16% of our randomly generated t-distribution values are greater than 1.

Chapter 5

Example 1: Let's return to the Hubble dataset. Let's load the dataset into R and then plot the data. Note the plot parameter **pch=16** fills in the data points for a nicer looking plot.

```
plot(Velocity~Distance,data=hubble, pch=16)
```



There appears to be a positive relationship between Distance and Velocity. We can calculate the correlation and covariance between the two variables as well.

```
cor(hubble$Distance,hubble$Velocity)

## [1] 0.789032

cov(hubble$Distance,hubble$Velocity)

## [1] 189.159
```

The correlation is +0.789 indicating a positive, moderately strong linear relationship between Distance and Velocity. The covariance is 189.159 which is less interpretable since covariance is dependent on the units of measurement. For this reason, we typically prefer to use correlation rather than covariance.

Example 2: The **mtcars** dataset is provided in the base version of R. To find out more info about this dataset, you can get details by calling **?mtcars**.

```
# get info on the mtcars dataset
?mtcars
```

In the side window of R Studio, it should display the following description after running **?mtcars**:
The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (197374 models).

Let's load the dataset and take a look at the first few values.

```
data(mtcars)
head(mtcars)

##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

In the code below, we extract just these two variables from **mtcars** and create a new variable **m**. From this new variable **m**, we can construct a two-way table of counts for the combinations of **cyl** and **gear**.

```
# extract the variables of interest
m = mtcars[,c("cyl","gear")]

# construct a two-way table of counts for cyl and gear
m.table = table(m); m.table

##    gear
## cyl  3  4  5
##   4  1  8  2
##   6  2  4  1
##   8 12  0  2
```

We can also construct the joint probability mass function by dividing our table by the total number of observations.

```
# calculate the number of total observations
n = sum(m.table); n

## [1] 32

# construct the joint pmf
m.pmf = table(m)/n; m.pmf

##    gear
## cyl       3       4       5
##   4 0.03125 0.25000 0.06250
##   6 0.06250 0.12500 0.03125
##   8 0.37500 0.00000 0.06250
```

Using the table, we can see that P(gear=3 and cyl=8) = 0.375 for example. What is P(gear=4 and cylinder = 4)? The middle entry of the table tells us this value is 0.125. In other words, 12.5% of the cars in this dataset have 4 gears and 4 cylinders.

Next, we are going to look at the joint probability mass function (PMF) of the variables **cyl** and **gear**. We can get the marginal PMFs by summing across rows and columns of the joint pmf.

```
# marginal pmf of gear
colSums(m.pmf)

##       3       4       5
## 0.46875 0.37500 0.15625

# marginal pmf of cyl
rowSums(m.pmf)

##       4       6       8
## 0.34375 0.21875 0.43750
```

From the tables above, you should see that P(gear=3)=0.46875 and P(cyl=4)=0.34375. Finally, we can get the correlation as follows. Note that gear and cyl have a negative, moderate linear relationship.

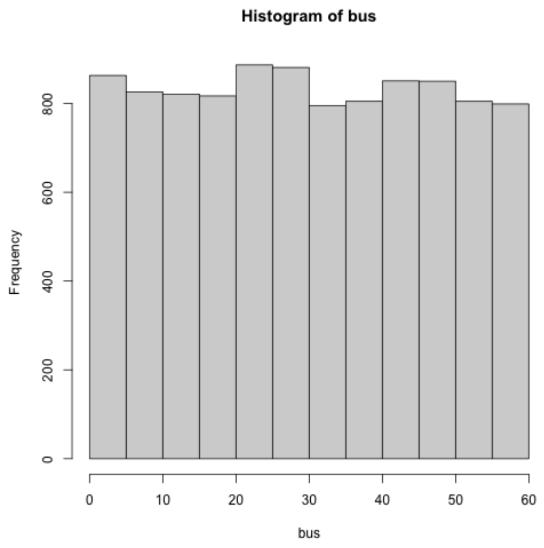
```
cor(m$cyl,m$gear)

## [1] -0.4926866
```

Chapter 6

Example 1: What does the distribution of the sample means of independent (continuous) uniform random variables look like? The Central Limit Theorem tells us that for a large sample size, the distribution of such means will be approximately normally distributed. Let's see this in action.

Suppose we know a bus arrives once per hour at a designated stop but we don't know when the next bus is and we don't know when the last bus came. We will model this as a continuous uniform[0,60] random variable. In other words, the next bus could come in anywhere from 0 to 60 minutes with all possibilities being equally likely. In the code below, we simulate 10,000 uniform[0,60] random variables and look at the histogram of results.

```
# Generate 10,000 uniform[0,60] random varibles
set.seed(2020)
bus = runif(10000,0,60)
hist(bus)
```



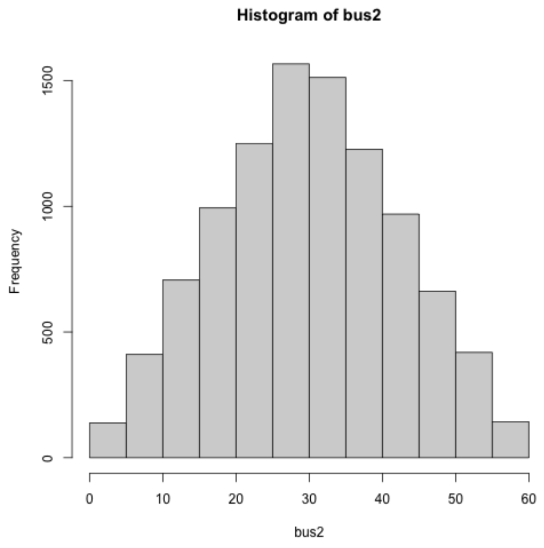
The mean and variance of this sampling distribution is calculated below.

```
mean(bus)

## [1] 29.79179

var(bus)

## [1] 298.1569
```

From chapter 4, we know that if $X \sim Uniform[a,b]$, then $E[X] = \frac{b+a}{2}$ and $Var(X) = \frac{(b-a)^2}{12}$. For a uniform[0,60] random variable, we get the theoretical values of $E[X] = \frac{60+0}{2} = 30$ and $Var(X) = \frac{(60-0)^2}{12} = 300$. Our simulated values are very close to these theoretical values.

Now, let's suppose we have to wait for two buses, each being modeled as an independent uniform[0,60]. Let's look at the sampling distribution for the mean of the two bus wait times.

```
set.seed(2020)
bus2 = rep(0, 10000) # initialize this vector to zeros

# 10,000 simulations of 2 bus waits
for (i in 1:10000){
x1 = runif(2,0,60) # n=2
bus2[i] = mean(x1)
}

mean(bus2)

## [1] 29.90972

var(bus2)

## [1] 149.6154

hist(bus2)
```

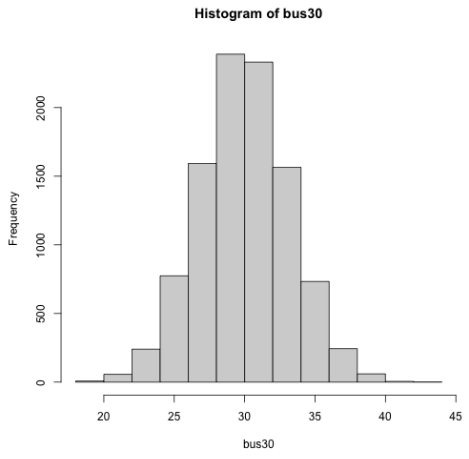


Notice that the average of two bus waits is no longer uniform and instead has more of a triangular shape. Our simulations of two bus waits had a mean of 30.05955 and variance of 151.2226. From chapter 6, we know that $E[\bar{X}] = \mu$ and $Var(\bar{X}) = \frac{\sigma^2}{n}$.

For this simulation, the theoretical values for these quantities are $E[\bar{X}] = \mu = 30$ and $Var(\bar{X}) = \frac{\sigma^2}{n} = \frac{300}{2} = 150$. Again, our simulations give values very close to the theoretical values.

Finally, let's suppose that we repeat the experiment for 30 days straight and look at the distribution of sample means of 30 days. We will simulate these 30 day periods 10,000 times.

```r
set.seed(2020)
bus30 = rep(0, 10000) # initialize this vector to zeros

# 10,000 simulations of 2 bus waits
for (i in 1:10000){
x2 = runif(30,0,60) # n=30
bus30[i] = mean(x2)
}

mean(bus30)

## [1] 29.97879

var(bus30)

## [1] 10.19619

hist(bus30)
```



Our simulations of 30 bus waits had a mean of 30.00598 and variance of 10.08975. The theoretical values for this simulation are $E[\bar{X}] = \mu = 30$ and $Var(\bar{X}) = \frac{\sigma^2}{n} = \frac{300}{30} = 10$.

Notice that the distribution of sample means is narrower for n=30 than n=2 and the variance is lower when a larger sample size is used. Also note that the distribution is somewhat bell-shaped.

In the coming chapters on Inferential Statistics, we will use the fact that the Central Limit Theorem states that the sampling distribution of sample means is approximately normally distributed.

Chapter 7

Example: The following example uses the dataset **faithful** which contains the duration of eruption (in minutes) and waiting times between eruptions for Old Faithful geyser in Yellowstone National Park, Wyoming, USA.

```r
# look up info on the faithful dataset
?faithful

# load the faithful dataset
data(faithful)
```

We can use the **t.test** function to run a hypothesis (see chapter 8) and to create a confidence interval for an unknown parameter.

Suppose we want to create a 95% confidence interval for the true mean duration of eruptions for Old Faithful. By default, R will produce a 95% confidence interval unless otherwise specified. Since the population variance of eruption time for Old Faithful is unknown, the confidence interval will be created using the t-distribution.

```r
t.test(faithful$eruptions)

##
##  One Sample t-test
##
## data:  faithful$eruptions
## t = 50.397, df = 271, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  3.351534 3.624032
## sample estimates:
## mean of x
##  3.487783
```

In the output, we see that the 95% confidence interval for the true mean eruption duration is $3.351534 < \mu < 3.624032$. If someone were to claim that the average eruption duration of Old Faithful is 4 minutes, we would reject this claim.

We could create a 99% confidence interval for the true mean eruption duration as well. This will be a wider confidence interval since we are requiring higher confidence.

```
t.test(faithful$eruptions,conf.level = 0.99)

##
##  One Sample t-test
##
## data:  faithful$eruptions
## t = 50.397, df = 271, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 99 percent confidence interval:
##  3.308257 3.667309
## sample estimates:
## mean of x
##  3.487783
```

The 99% confidence interval for the true mean eruption duration is $3.308257 < \mu < 3.667309$. We can also create one-sided confidence intervals by using the **alternative** argument with either **greater** or less. Below, a 90% upper one-sided confidence interval is created.

```
t.test(faithful$eruptions,conf.level = 0.90,alternative = "greater")

##
##  One Sample t-test
##
## data:  faithful$eruptions
## t = 50.397, df = 271, p-value < 2.2e-16
## alternative hypothesis: true mean is greater than 0
## 90 percent confidence interval:
##  3.398876      Inf
## sample estimates:
## mean of x
##  3.487783
```

The 90% upper one-sided confidence interval is $3.398876 < \mu$. We are 90% confident that the true mean eruption duration of Old Faithful is at least 3.339 minutes.

Chapter 8

Example: The **faithful** dataset of Old Faith Geyser data is used again for this example. Since the population variance of Old Faithful eruption duration is unknown, the t-distribution is used to complete the following hypothesis tests.

Let μ denote the true mean eruption duration of Old Faithful and we want to test $H_0 : \mu = 3$ vs. $H_a : \mu \neq 3$ at $\alpha = 0.05$.

```
t.test(faithful$eruptions,mu = 3)

##
##  One Sample t-test
##
## data:  faithful$eruptions
## t = 7.0483, df = 271, p-value = 1.502e-11
## alternative hypothesis: true mean is not equal to 3
## 95 percent confidence interval:
##  3.351534 3.624032
## sample estimates:
## mean of x
##  3.487783
```

From the output, we see that the p-value is $1.5 \cdot 10^{-11}$. This is less than $\alpha = 0.05$, so we would reject $H_0 : \mu = 3$ and conclude the true mean eruption duration of Old Faithful differs from 3 minutes.

Suppose we want to test $H_0 : \mu = 3.5$ vs. $H_a : \mu < 3.5$ at $\alpha = 0.05$. In other words, we would like to test to see if the mean is less than 3.5 minutes.

```
t.test(faithful$eruptions,mu=3.5,alternative = "less")

##
##  One Sample t-test
##
## data:  faithful$eruptions
## t = -0.17653, df = 271, p-value = 0.43
## alternative hypothesis: true mean is less than 3.5
## 95 percent confidence interval:
##      -Inf 3.602007
## sample estimates:
## mean of x
##  3.487783
```

Here, the p-value is 0.43, so we would fail to reject $H_0 : \mu = 3.5$. In other words, we don't have sufficient evidence to conclude the true mean eruption duration is less than 3.5 minutes.

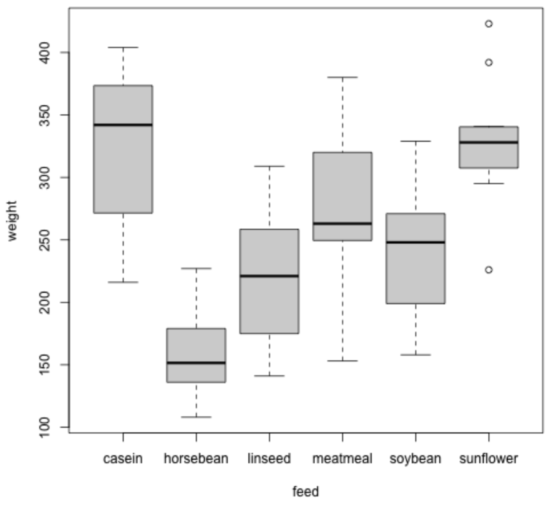
Chapter 9

Example: The dataset **chickwts** is included in the base version of R. This dataset include 71 observations of the growth rate of chickens using a variety of feed supplements.

```
data(chickwts)

# pick out some random indices of the dataset and look at the elements
set.seed(2020)
chickwts[sample(x=71,size=10,replace = F),]

##    weight      feed
## 28    250   soybean
## 22    271   linseed
## 65    318    casein
## 17    148   linseed
## 36    248   soybean
## 42    226 sunflower
## 49    325  meatmeal
## 45    334 sunflower
## 56    242  meatmeal
## 8     124 horsebean
```

We can create a boxplot of the weights for the various types of feed.

```
boxplot(weight~feed,data=chickwts)
```



Based on the boxplot, there appears be a difference in weights between **casein** and **horsebean**. To test this, let's run the hypothesis test $H_0 : \mu_1 - \mu_2 = 0$ vs. $H_a : \mu_1 - \mu_2 \neq 0$, where μ_1 is the true mean weight of chicks given casein and μ_2 is the true mean weight of chicks given horsebean. We will use $\alpha = 0.05$ for the following examples.

```
# create new dataset of chicks given casein
chicks1 = chickwts[chickwts$feed=="casein",]
chicks2 = chickwts[chickwts$feed=="horsebean",]
t.test(chicks1$weight,chicks2$weight,mu=0)

##
##  Welch Two Sample t-test
##
## data:  chicks1$weight and chicks2$weight
## t = 7.3423, df = 18.36, p-value = 7.21e-07
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  116.6982 210.0685
## sample estimates:
## mean of x mean of y
##  323.5833  160.2000
```

The test statistic is $t = 7.34$ and corresponds to p-value of $7.21 \cdot 10^{-7}$. Since the p-value is less than $\alpha = 0.05$, we would reject H_0 and conclude that the true mean chick weights differ between casein and horsebean feed.

Suppose we want to test if the true mean weight of casein feed chicks is more than 200 units greater than the true mean weight of horsebean feed chicks, i.e., $H_0 : \mu_1 - \mu_2 = 200$ vs. $H_a : \mu_1 - \mu_2 > 200$.

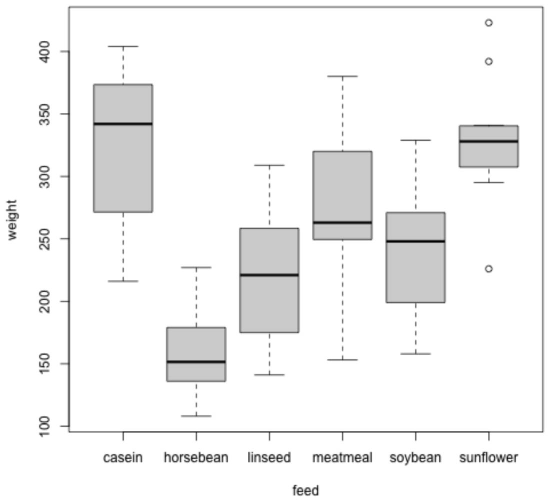
```
t.test(chicks1$weight,chicks2$weight,mu=200,alternative = "greater")

##
##  Welch Two Sample t-test
##
## data:  chicks1$weight and chicks2$weight
## t = -1.6455, df = 18.36, p-value = 0.9416
## alternative hypothesis: true difference in means is greater than 200
## 95 percent confidence interval:
##  124.8371      Inf
## sample estimates:
## mean of x mean of y
##  323.5833  160.2000
```

The test statistic for this hypothesis test is $t = -1.6455$ with a p-value of 0.9416. At $\alpha = 0.05$, we would fail to reject $H_0 : \mu_1 - \mu_2 = 200$ and conclude there is not sufficient evidence to say that the true mean weight of casein feed chicks is more than 200 units greater than the true mean weight of horsebean feed chicks.

Chapter 10

Example: This example uses the **chickwts** dataset again. This dataset include 71 observations of the growth rate of chickens using a variety of feed supplements. The boxplot is produced again below.

```r
data(chickwts)
boxplot(weight~feed,data=chickwts)
```



Using the theory from Chapter 10, we are ready to test if there is evidence of a difference in mean weights of chicks for the six types of feed. Our ANOVA F-test hypotheses will be:
$H_0 : \mu_1 = \mu_2 = \mu_3 = \mu_4 = \mu_5 = \mu_6$ vs. H_a : at least one μ_i differs ($i = 1, 2, 3, 4, 5, 6$).

Let's run this F-test at $\alpha = 0.01$.

```r
fit = aov(weight~feed,data=chickwts)
summary(fit)

##             Df Sum Sq Mean Sq F value   Pr(>F)
## feed         5 231129   46226   15.37 5.94e-10 ***
## Residuals   65 195556    3009
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The resulting F-test statistic is $F_{test} = 15.37$ ($df_1 = 5, df_2 = 65$) and a p-value of $5.94 \cdot 10^{-10}$. At a significance level of $\alpha = 0.01$, we would reject H_0 and conclude that at least one of the true mean weights differs by diet.

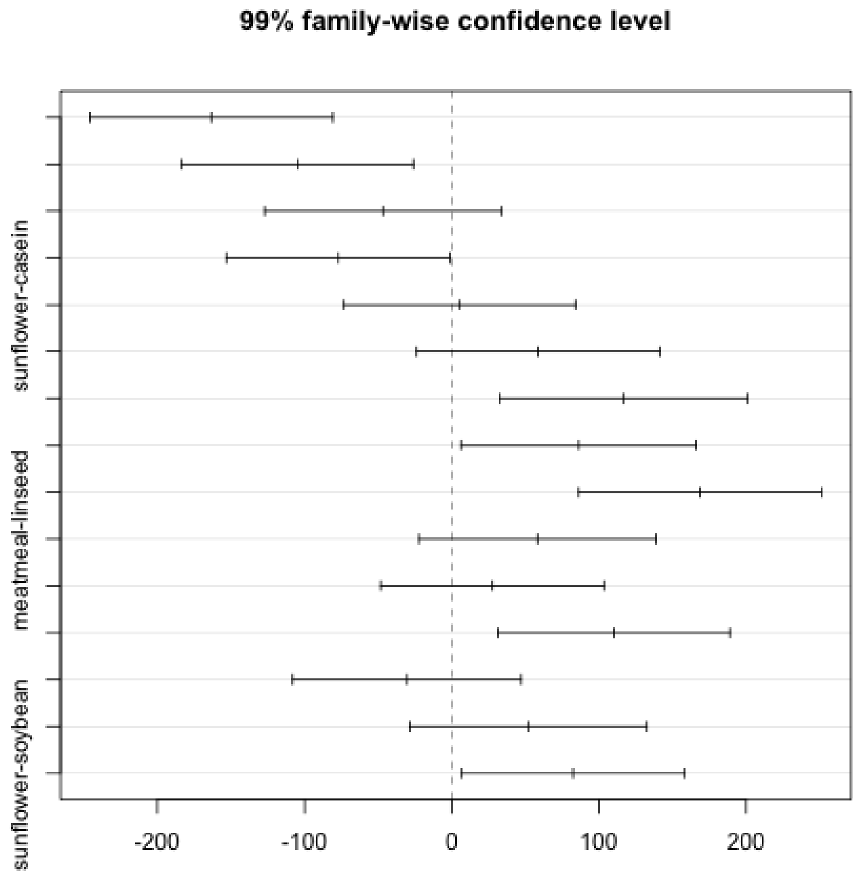
Which μ_i differ? We can answer this by running all pairwise comparisons of μ_i and μ_j ($i \neq j$) and making a multiple comparisons adjustment. Let's use a family-wise error rate of $\alpha = 0.01$.

```
TukeyHSD(fit,conf.level = 0.99)

##   Tukey multiple comparisons of means
##     99% family-wise confidence level
##
## Fit: aov(formula = weight ~ feed, data = chickwts)
##
## $feed
##                            diff         lwr        upr     p adj
## horsebean-casein    -163.383333 -245.964363 -80.802304 0.0000000
## linseed-casein      -104.833333 -183.571256 -26.095411 0.0002100
## meatmeal-casein      -46.674242 -127.181777  33.833292 0.3324584
## soybean-casein       -77.154762 -153.028522  -1.281001 0.0083653
## sunflower-casein       5.333333  -73.404589  84.071256 0.9998902
## linseed-horsebean     58.550000  -24.031030 141.131030 0.1413329
## meatmeal-horsebean   116.709091   32.439113 200.979069 0.0001062
## soybean-horsebean     86.228571    6.373743 166.083400 0.0042167
## sunflower-horsebean  168.716667   86.135637 251.297696 0.0000000
## meatmeal-linseed      58.159091  -22.348444 138.666626 0.1276965
## soybean-linseed       27.678571  -48.195189 103.552332 0.7932853
## sunflower-linseed    110.166667   31.428744 188.904589 0.0000884
## soybean-meatmeal     -30.480519 -108.189144  47.228105 0.7391356
## sunflower-meatmeal    52.007576  -28.499959 132.515111 0.2206962
## sunflower-soybean     82.488095    6.614335 158.361856 0.0038845
```

There are several statistically significant differences in mean chick weights at $\alpha = 0.01$.

Casein differs from horsebean, linseed, and soybean.
Horsebean differs from casein, meatmeal, soybean, and sunflower.
Linseed differs from casein and sunflower.
Meatmeal differs from horsebean.
Sunflower differs from horsebean, linseed, and soybean.
Soybean differs from sunflower, horsebean, and casein.

```r
plot(TukeyHSD(fit,conf.level = 0.99))
```



Chapter 11

Example: The **pressure** dataset is provided in the base version of R. The data is on the relation between temperature in degrees Celsius and vapor pressure of mercury in millimeters (of mercury).

Let's begin by looking at the header of the data and calculating the correlation.

```
data(pressure)
head(pressure)

##   temperature pressure
## 1           0   0.0002
## 2          20   0.0012
## 3          40   0.0060
## 4          60   0.0300
## 5          80   0.0900
## 6         100   0.2700

cor(pressure)

##             temperature  pressure
## temperature   1.0000000 0.7577923
## pressure      0.7577923 1.0000000
```

The correlation matrix is given in the example above. The diagonal elements of the correlation matrix shows that any variable is perfectly correlated with itself (i.e., $r_{xx} = 1$). In the off diagonal, you can see the correlation between the **temperature** and **pressure** is 0.758 which a strong, positive linear relationship.

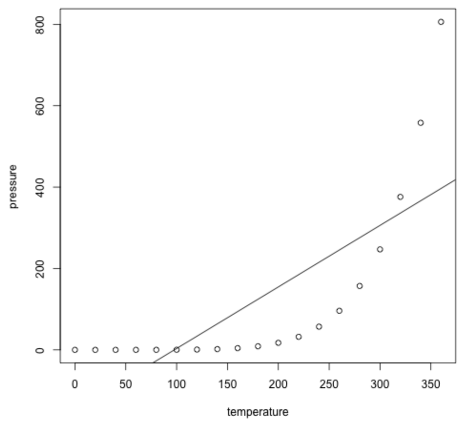
Let's fit a simple linear regression model with the **temperature** being the predictor variable and **pressure** being the response variable.

```
model = lm(pressure~temperature,data=pressure)
summary(model)

##
## Call:
## lm(formula = pressure ~ temperature, data = pressure)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -158.08 -117.06  -32.84   72.30  409.43
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -147.8989    66.5529  -2.222 0.040124 *
## temperature    1.5124     0.3158   4.788 0.000171 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 150.8 on 17 degrees of freedom
## Multiple R-squared:  0.5742,Adjusted R-squared:  0.5492
## F-statistic: 22.93 on 1 and 17 DF,  p-value: 0.000171
```

The estimated regression is $\widehat{pressure} = -147.90 + 1.51 \cdot temperature$. A one degree Celsius increase in temperate of mercury is associated with an increase of 1.51 in vapor pressure.

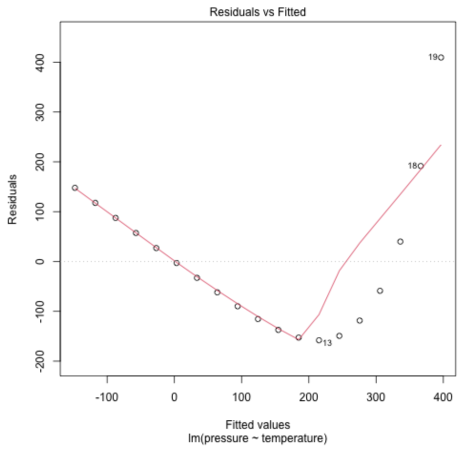
Is this model appropriate? Let's plot the data along with the line of best fit.

```
plot(pressure~temperature,data=pressure)
abline(model)
```



As seen in the above plot, our simple linear regression model is completely inadequate. We could determine that a nonlinearity exists by looking at the first residual diagnostic plot.

```
plot(model,which=1)
```

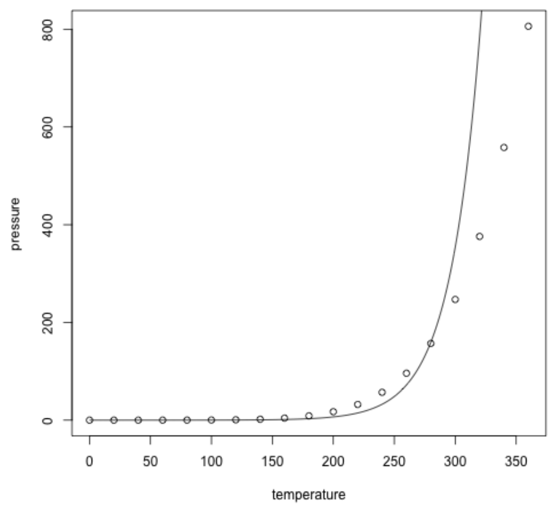


In this diagnostic plot, we see strong evidence of an unaccounted for nonlinearity as the shape of the graph is not approximately flat.

Based on the scatterplot on the previous page, there appears to be a possibly exponential relationship between temperature and pressure. Let's try log transforming pressure and refitting the linear regression model. (Note: Variable transformations are covered in Chapter 12.)

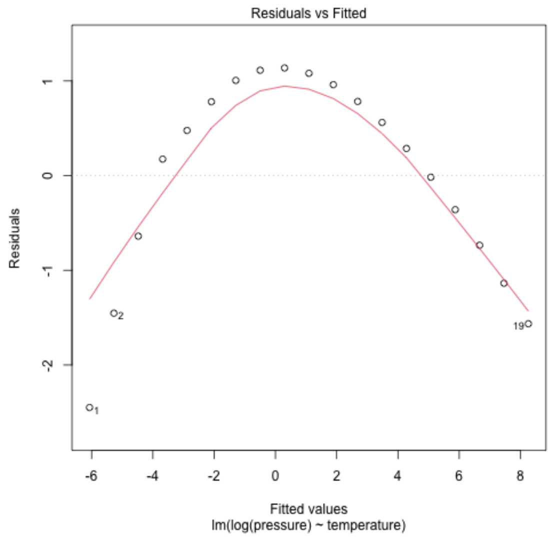
```
model2 = lm(log(pressure)~temperature,data=pressure)
summary(model2)

##
## Call:
## lm(formula = log(pressure) ~ temperature, data = pressure)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.4491 -0.6876  0.2866  0.8716  1.1365
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -6.068144   0.483831  -12.54 5.10e-10 ***
## temperature  0.039792   0.002296   17.33 3.07e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.096 on 17 degrees of freedom
## Multiple R-squared:  0.9464,Adjusted R-squared:  0.9433
## F-statistic: 300.3 on 1 and 17 DF,  p-value: 3.07e-12
```

```r
plot(pressure~temperature,data=pressure)

x.new = 0:350
y.new = predict(model2,newdata=list(temperature=x.new,interval="confidence"))
lines(x.new,exp(y.new))
```



This model seems to fit the data better than the untransformed simple linear regression model but is still somewhat inadequate for temperatures greater than 300C. Additionally, the residual diagnostic plot suggest our model is inadequate. A better model should be found if modeling temperatures above 300C is of interest.

```r
plot(model2,which=1)
```



Chapter 12

Example: The **longley** dataset is provide in the base version of R and is a macroeconomic data set which provides a well-known example for a highly collinear regression. This dataset includes 7 economical variables, observed yearly from 1947 to 1962 (n=16). For more information on the variables, enter **?longley** into R.

Let's begin by looking at the header of the dataset and calculating the correlation matrix.

```
data(longley)

# show first few observations
head(longley)

##      GNP.deflator     GNP Unemployed Armed.Forces Population Year Employed
## 1947         83.0 234.289      235.6        159.0    107.608 1947   60.323
## 1948         88.5 259.426      232.5        145.6    108.632 1948   61.122
## 1949         88.2 258.054      368.2        161.6    109.773 1949   60.171
## 1950         89.5 284.599      335.1        165.0    110.929 1950   61.187
## 1951         96.2 328.975      209.9        309.9    112.075 1951   63.221
## 1952         98.1 346.999      193.2        359.4    113.270 1952   63.639

# round the correlations to two decimal places for better viewing
round(cor(longley),2)

##              GNP.deflator  GNP Unemployed Armed.Forces Population Year Employed
## GNP.deflator         1.00 0.99       0.62         0.46       0.98 0.99     0.97
## GNP                  0.99 1.00       0.60         0.45       0.99 1.00     0.98
## Unemployed           0.62 0.60       1.00        -0.18       0.69 0.67     0.50
## Armed.Forces         0.46 0.45      -0.18         1.00       0.36 0.42     0.46
## Population           0.98 0.99       0.69         0.36       1.00 0.99     0.96
## Year                 0.99 1.00       0.67         0.42       0.99 1.00     0.97
## Employed             0.97 0.98       0.50         0.46       0.96 0.97     1.00
```

For this example, we will use **Employed** (number of people employed) as the response variable and the other six variables as predictor variables.

Let's begin by using all six predictor variables and fitting the multiple regression model.

```
model1 = lm(Employed~GNP.deflator+GNP+Unemployed+Armed.Forces+Population+Year,
           data=longley)
summary(model1)

##
## Call:
## lm(formula = Employed ~ GNP.deflator + GNP + Unemployed + Armed.Forces +
##     Population + Year, data = longley)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.41011 -0.15767 -0.02816  0.10155  0.45539
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -3.482e+03  8.904e+02  -3.911 0.003560 **
## GNP.deflator  1.506e-02  8.492e-02   0.177 0.863141
## GNP          -3.582e-02  3.349e-02  -1.070 0.312681
## Unemployed   -2.020e-02  4.884e-03  -4.136 0.002535 **
## Armed.Forces -1.033e-02  2.143e-03  -4.822 0.000944 ***
## Population   -5.110e-02  2.261e-01  -0.226 0.826212
## Year          1.829e+00  4.555e-01   4.016 0.003037 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3049 on 9 degrees of freedom
## Multiple R-squared:  0.9955,Adjusted R-squared:  0.9925
## F-statistic: 330.3 on 6 and 9 DF,  p-value: 4.984e-10
```

As can be seen in the above output, three variables (**GNP.deflator**, **GNP**, and **Population**) are not significant.

Let's take a stepwise approach, remove the variable with the largest p-value that is not statistically significant, and refit the model. In this case, let's remove the predictor variable **GNP.deflator** and refit the multiple regression model.

```
model2 = lm(Employed~GNP+Unemployed+Armed.Forces+Population+Year,
           data=longley)
summary(model2)

##
## Call:
## lm(formula = Employed ~ GNP + Unemployed + Armed.Forces + Population +
##     Year, data = longley)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.43015 -0.15399 -0.01832  0.10081  0.44964
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -3.450e+03  8.282e+02  -4.165 0.001932 **
## GNP          -3.196e-02  2.420e-02  -1.321 0.216073
## Unemployed   -1.972e-02  3.861e-03  -5.108 0.000459 ***
## Armed.Forces -1.020e-02  1.908e-03  -5.345 0.000326 ***
## Population   -7.754e-02  1.616e-01  -0.480 0.641607
## Year          1.814e+00  4.253e-01   4.266 0.001648 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2897 on 10 degrees of freedom
## Multiple R-squared:  0.9955,Adjusted R-squared:  0.9932
## F-statistic: 438.8 on 5 and 10 DF,  p-value: 2.242e-11
```

The **Population** variable is again statistically not significant and has the largest p-value, so let's remove that from the model.

```
model3 = lm(Employed~GNP+Unemployed+Armed.Forces+Year,
           data=longley)
summary(model3)

##
## Call:
## lm(formula = Employed ~ GNP + Unemployed + Armed.Forces + Year,
##     data = longley)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.42165 -0.12457 -0.02416  0.08369  0.45268
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -3.599e+03  7.406e+02  -4.859 0.000503 ***
## GNP          -4.019e-02  1.647e-02  -2.440 0.032833 *
## Unemployed   -2.088e-02  2.900e-03  -7.202 1.75e-05 ***
## Armed.Forces -1.015e-02  1.837e-03  -5.522 0.000180 ***
## Year          1.887e+00  3.828e-01   4.931 0.000449 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2794 on 11 degrees of freedom
## Multiple R-squared:  0.9954,Adjusted R-squared:  0.9937
## F-statistic: 589.8 on 4 and 11 DF,  p-value: 9.5e-13
```

All of the predictor variables are now statistically significant at $\alpha = 0.05$.

Note that $R^2 = 0.9954$, so 99.54% of the variability in the number of employed people can be explained by the linear relationship with gross national product (GNP), number of unemployed people, number of people in the armed forces, and year. This is a very large value for R^2 which suggests a very good model fit.

One caution is that our predictor variables are highly correlated (see correlation matrix at the beginning of the example), so this could result in high standard errors. This concern of **multicollinearity** is beyond the scope of this class and is covered in more advanced statistics courses.