STA 578 - Statistical Computing Notes

Derek Sonderegger

2017-10-15

Contents

Preface

This is a set of lecture notes that I will use for Northern Arizona University's STA 578 course titled "Statistical Computing". For this course we will primarily be using the textbook "An Introduction to Statistical Learning" by James, Witten, Hastie, and Tibshirani. As such, the notes will primarily be to supplement the text book and these shouldn't be considered comprehensive, stand-alone notes.

Chapter 1

Data Manipulation

Most of the time, our data is in the form of a data frame and we are interested in exploring the relationships. This chapter explores how to manipulate data frames and methods.

1.1 Classic R functions for summarizing rows and columns

1.1.1 `summary()`

The first method is to calculate some basic summary statistics (minimum, 25th, 50th, 75th percentiles, maximum and mean) of each column. If a column is categorical, the summary function will return the number of observations in each category.

```
# use the iris data set which has both numerical and categorical variables
data( iris )
str(iris)    # recall what columns we have
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
# display the summary for each column
summary( iris )
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
```

##

1.1.2 apply()

The summary function is convenient, but we want the ability to pick another function to apply to each column and possibly to each row. To demonstrate this, suppose we have data frame that contains students grades over the semester.

```
# make up some data
grades <- data.frame(
  l.name = c('Cox', 'Dorian', 'Kelso', 'Turk'),
  Exam1 = c(93, 89, 80, 70),
  Exam2 = c(98, 70, 82, 85),
  Final = c(96, 85, 81, 92) )
```

The `apply()` function will apply an arbitrary function to each row (or column) of a matrix or a data frame and then aggregate the results into a vector.

```
# Because I can't take the mean of the last names column,
# remove the name column
scores <- grades[,-1]
scores
```

```
##   Exam1 Exam2 Final
## 1    93    98    96
## 2    89    70    85
## 3    80    82    81
## 4    70    85    92
```

```
# Summarize each column by calculating the mean.
apply( scores,      # what object do I want to apply the function to
       MARGIN=2,    # rows = 1, columns = 2, (same order as [rows, cols]
       FUN=mean     # what function do we want to apply
     )
```

```
## Exam1 Exam2 Final
## 83.00 83.75 88.50
```

To apply a function to the rows, we just change which margin we want. We might want to calculate the average exam score for person.

```
apply( scores,      # what object do I want to apply the function to
       MARGIN=1,    # rows = 1, columns = 2, (same order as [rows, cols]
       FUN=mean     # what function do we want to apply
     )
```

```
## [1] 95.66667 81.33333 81.00000 82.33333
```

This is useful, but it would be more useful to concatenate this as a new column in my grades data frame.

```
average <- apply(
  scores,      # what object do I want to apply the function to
  MARGIN=1,    # rows = 1, columns = 2, (same order as [rows, cols]
  FUN=mean     # what function do we want to apply
)
grades <- cbind( grades, average ) # squish together
grades
```

```
##   l.name Exam1 Exam2 Final  average
## 1    Cox    93    98    96 95.66667
## 2 Dorian    89    70    85 81.33333
## 3  Kelso    80    82    81 81.00000
## 4   Turk    70    85    92 82.33333
```

There are several variants of the `apply()` function, and the variant I use most often is the function `sapply()`, which will apply a function to each element of a list or vector and returns a corresponding list or vector of results.

1.2 Package `dplyr`

```
library(dplyr)   # load the dplyr package!
```

Many of the tools to manipulate data frames in R were written without a consistent syntax and are difficult use together. To remedy this, Hadley Wickham (the writer of `ggplot2`) introduced a package called plyr which was quite useful. As with many projects, his first version was good but not great and he introduced an improved version that works exclusively with data.frames called `dplyr` which we will investigate. The package `dplyr` strives to provide a convenient and consistent set of functions to handle the most common data frame manipulations and a mechanism for chaining these operations together to perform complex tasks.

The Dr Wickham has put together a very nice introduction to the package that explains in more detail how the various pieces work and I encourage you to read it at some point. [http://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html].

One of the aspects about the `data.frame` object is that R does some simplification for you, but it does not do it in a consistent manner. Somewhat obnoxiously character strings are always converted to factors and subsetting might return a `data.frame` or a `vector` or a `scalar`. This is fine at the command line, but can be problematic when programming. Furthermore, many operations are pretty slow using `data.frame`. To get around this, Dr Wickham introduced a modified version of the `data.frame` called a `tibble`. A `tibble` is a `data.frame` but with a few extra bits. For now we can ignore the differences.

The pipe command `%>%` allows for very readable code. The idea is that the `%>%` operator works by translating the command `a %>% f(b)` to the expression `f(a,b)`. This operator works on any function and was introduced in the `magrittr` package. The beauty of this comes when you have a suite of functions that takes input arguments of the same type as their output.

For example if we wanted to start with `x`, and first apply function `f()`, then `g()`, and then `h()`, the usual R command would be `h(g(f(x)))` which is hard to read because you have to start reading at the innermost set of parentheses. Using the pipe command `%>%`, this sequence of operations becomes `x %>% f() %>% g() %>% h()`.

Dr Wickham gave the following example of readability:

```
bopping(
  scooping_up(
    hopping_through(foo_foo),
    field_mice),
  head)
```

is more readably written:

```
foo_foo %>%
  hopping_through(forest) %>%
  scooping_up( field_mice) %>%
  bopping( head )
```

In `dplyr`, all the functions below take a data set as its first argument and outputs an appropriately modified data set. This will allow me to chain together commands in a readable fashion.

1.2.1 Verbs

The foundational operations to perform on a data set are:

- Subsetting - Returns a with only particular columns or rows
  - `select` - Selecting a subset of columns by name or column number.
  - `filter` - Selecting a subset of rows from a data frame based on logical expressions.
  - `slice` - Selecting a subset of rows by row number.
- `arrange` - Re-ordering the rows of a data frame.
- `mutate` - Add a new column that is some function of other columns.
- `summarise` - calculate some summary statistic of a column of data. This collapses a set of rows into a single row.

Each of these operations is a function in the package `dplyr`. These functions all have a similar calling syntax, the first argument is a data set, subsequent arguments describe what to do with the input data frame and you can refer to the columns without using the `df$column` notation. All of these functions will return a data set.

1.2.1.1 Subsetting with `select`, `filter`, and `slice`

These function allows you select certain columns and rows of a data frame.

1.2.1.1.1 `select()`

Often you only want to work with a small number of columns of a data frame. It is relatively easy to do this using the standard `[,col.name]` notation, but is often pretty tedious.

```
# recall what the grades are
grades
```

```
##   l.name Exam1 Exam2 Final  average
## 1    Cox    93    98    96 95.66667
## 2 Dorian    89    70    85 81.33333
## 3  Kelso    80    82    81 81.00000
## 4   Turk    70    85    92 82.33333
```

I could select the columns Exam columns by hand, or by using an extension of the `:` operator

```
grades %>% select( Exam1, Exam2 )   # Exam1 and Exam2
```

```
##   Exam1 Exam2
## 1    93    98
## 2    89    70
## 3    80    82
## 4    70    85
```

```
grades %>% select( Exam1:Final )    # Columns Exam1 through Final
```

```
##   Exam1 Exam2 Final
## 1    93    98    96
## 2    89    70    85
## 3    80    82    81
## 4    70    85    92
```

```
grades %>% select( -Exam1 )          # Negative indexing by name works
```

```
##   l.name Exam2 Final  average
## 1    Cox    98    96 95.66667
## 2 Dorian    70    85 81.33333
## 3  Kelso    82    81 81.00000
## 4   Turk    85    92 82.33333
```

```
grades %>% select( 1:2 )             # Can select column by column position
```

```
##   l.name Exam1
## 1    Cox    93
## 2 Dorian    89
## 3  Kelso    80
## 4   Turk    70
```

The `select()` command has a few other tricks. There are functional calls that describe the columns you wish to select that take advantage of pattern matching. I generally can get by with `starts_with()`, `ends_with()`, and `contains()`, but there is a final operator `matches()` that takes a regular expression.

```
grades %>% select( starts_with('Exam') )    # Exam1 and Exam2
```

```
##   Exam1 Exam2
## 1    93    98
## 2    89    70
## 3    80    82
## 4    70    85
```

1.2.1.1.2 `filter()`

It is common to want to select particular rows where we have some logically expression to pick the rows.

```
# select students with Final grades greater than 90
grades %>% filter(Final > 90)
```

```
##   l.name Exam1 Exam2 Final  average
## 1    Cox    93    98    96 95.66667
## 2   Turk    70    85    92 82.33333
```

You can have multiple logical expressions to select rows and they will be logically combined so that only rows that satisfy all of the conditions are selected. The logicals are joined together using `&` (and) operator or the `|` (or) operator and you may explicitly use other logicals. For example a factor column type might be used to select rows where type is either one or two via the following: `type==1 | type==2`.

```
# select students with Final grades above 90 and
# average score also above 90
grades %>% filter(Final > 90, average > 90)
```

```
##   l.name Exam1 Exam2 Final  average
## 1    Cox    93    98    96 95.66667
```

```
# we could also use an "and" condition
grades %>% filter(Final > 90 & average > 90)
```

```
##   l.name Exam1 Exam2 Final  average
## 1    Cox    93    98    96 95.66667
```

1.2.1.1.3 slice()

When you want to filter rows based on row number, this is called slicing.

```
# grab the first 2 rows
grades %>% slice(1:2)
```

```
## # A tibble: 2 x 5
##   l.name Exam1 Exam2 Final  average
##   <fctr> <dbl> <dbl> <dbl>    <dbl>
## 1    Cox    93    98    96 95.66667
## 2 Dorian    89    70    85 81.33333
```

1.2.1.2 arrange()

We often need to re-order the rows of a data frame. For example, we might wish to take our grade book and sort the rows by the average score, or perhaps alphabetically. The `arrange()` function does exactly that. The first argument is the data frame to re-order, and the subsequent arguments are the columns to sort on. The order of the sorting column determines the precedent… the first sorting column is first used and the second sorting column is only used to break ties.

```
grades %>% arrange(l.name)
```

```
##   l.name Exam1 Exam2 Final  average
## 1    Cox    93    98    96 95.66667
## 2 Dorian    89    70    85 81.33333
## 3  Kelso    80    82    81 81.00000
## 4   Turk    70    85    92 82.33333
```

The default sorting is in ascending order, so to sort the grades with the highest scoring person in the first row, we must tell arrange to do it in descending order using `desc(column.name)`.

```
grades %>% arrange(desc(Final))
```

```
##   l.name Exam1 Exam2 Final  average
## 1    Cox    93    98    96 95.66667
## 2   Turk    70    85    92 82.33333
## 3 Dorian    89    70    85 81.33333
## 4  Kelso    80    82    81 81.00000
```

In a more complicated example, consider the following data and we want to order it first by Treatment Level and secondarily by the y-value. I want the Treatment level in the default ascending order (Low, Medium, High), but the y variable in descending order.

```
# make some data
dd <- data.frame(
  Trt = factor(c("High", "Med", "High", "Low"),
               levels = c("Low", "Med", "High")),
  y = c(8, 3, 9, 9),
  z = c(1, 1, 1, 2))
dd
```

```
##    Trt y z
## 1 High 8 1
```

```
## 2  Med 3 1
## 3 High 9 1
## 4  Low 9 2
```

```
# arrange the rows first by treatment, and then by y (y in descending order)
dd %>% arrange(Trt, desc(y))
```

```
##    Trt y z
## 1  Low 9 2
## 2  Med 3 1
## 3 High 9 1
## 4 High 8 1
```

1.2.1.3 mutate()

I often need to create a new column that is some function of the old columns. This was often cumbersome. Consider code to calculate the average grade in my grade book example.

```
grades$average <- (grades$Exam1 + grades$Exam2 + grades$Final) / 3
```

Instead, we could use the `mutate()` function and avoid all the `grades$` nonsense.

```
grades %>% mutate( average = (Exam1 + Exam2 + Final)/3 )
```

```
##   l.name Exam1 Exam2 Final  average
## 1    Cox    93    98    96 95.66667
## 2 Dorian    89    70    85 81.33333
## 3  Kelso    80    82    81 81.00000
## 4   Turk    70    85    92 82.33333
```

You can do multiple calculations within the same `mutate()` command, and you can even refer to columns that were created in the same `mutate()` command.

```
grades %>% mutate(
  average = (Exam1 + Exam2 + Final)/3,
  grade = cut(average, c(0, 60, 70, 80, 90, 100),  # cut takes numeric variable
                       c( 'F','D','C','B','A')) )  # and makes a factor
```

```
##   l.name Exam1 Exam2 Final  average grade
## 1    Cox    93    98    96 95.66667     A
## 2 Dorian    89    70    85 81.33333     B
## 3  Kelso    80    82    81 81.00000     B
## 4   Turk    70    85    92 82.33333     B
```

1.2.1.4 summarise()

By itself, this function is quite boring, but will become useful later on. Its purpose is to calculate summary statistics using any or all of the data columns. Notice that we get to chose the name of the new column. The way to think about this is that we are collapsing information stored in multiple rows into a single row of values.

```
# calculate the mean of exam 1
grades %>% summarise( mean.E1=mean(Exam1))
```

```
##   mean.E1
## 1      83
```

We could calculate multiple summary statistics if we like.

```
# calculate the mean and standard deviation
grades %>% summarise( mean.E1=mean(Exam1), stddev.E1=sd(Exam2) )
```

```
##   mean.E1 stddev.E1
## 1      83      11.5
```

If we want to apply the same statistic to each column, we use the `summarise_each()` command. We have to be a little careful here because the function you use has to work on every column (that isn't part of the grouping structure (see `group_by()`)).

```
# calculate the mean and stddev of each column - Cannot do this to Names!
grades %>%
  select( Exam1:Final ) %>%
  summarise_each( funs(mean, sd) )
```

```
## `summarise_each()` is deprecated.
## Use `summarise_all()`, `summarise_at()` or `summarise_if()` instead.
## To map `funs` over all variables, use `summarise_all()`
```

```
##   Exam1_mean Exam2_mean Final_mean Exam1_sd Exam2_sd Final_sd
## 1         83      83.75       88.5 10.23067     11.5 6.757712
```

1.2.1.5 Miscellaneous functions

There are some more function that are useful but aren't as commonly used. For sampling the functions `sample_n()` and `sample_frac()` will take a subsample of either n rows or of a fraction of the data set. The function `n()` returns the number of rows in the data set. Finally `rename()` will rename a selected column.

1.2.2 Split, apply, combine

Aside from unifying the syntax behind the common operations, the major strength of the `dplyr` package is the ability to split a data frame into a bunch of sub-dataframes, apply a sequence of one or more of the operations we just described, and then combine results back together. We'll consider data from an experiment from spinning wool into yarn. This experiment considered two different types of wool (A or B) and three different levels of tension on the thread. The response variable is the number of breaks in the resulting yarn. For each of the 6 `wool:tension` combinations, there are 9 replicated observations per `wool:tension` level.

```
data(warpbreaks)
str(warpbreaks)
```

```
## 'data.frame':    54 obs. of  3 variables:
##  $ breaks : num  26 30 54 25 70 52 51 26 67 18 ...
##  $ wool   : Factor w/ 2 levels "A","B": 1 1 1 1 1 1 1 1 1 1 ...
##  $ tension: Factor w/ 3 levels "L","M","H": 1 1 1 1 1 1 1 1 1 2 ...
```

The first we must do is to create a data frame with additional information about how to break the data into sub-dataframes. In this case, I want to break the data up into the 6 wool-by-tension combinations. Initially we will just figure out how many rows are in each wool-by-tension combination.

```
# group_by:  what variable(s) shall we group one
# n() is a function that returns how many rows are in the
#   currently selected sub-dataframe
warpbreaks %>%
  group_by( wool, tension) %>%    # grouping
  summarise(n = n() )             # how many in each group
```

```
## # A tibble: 6 x 3
## # Groups:   wool [?]
##     wool tension     n
##   <fctr>  <fctr> <int>
## 1      A       L     9
## 2      A       M     9
## 3      A       H     9
## 4      B       L     9
## 5      B       M     9
## 6      B       H     9
```

The `group_by` function takes a data.frame and returns the same data.frame, but with some extra information so that any subsequent function acts on each unique combination defined in the `group_by`. If you wish to remove this behavior, use `group_by()` to reset the grouping to have no grouping variable.

Using the same `summarise` function, we could calculate the group mean and standard deviation for each wool-by-tension group.

```
warpbreaks %>%
  group_by(wool, tension) %>%
  summarise( n           = n(),             # I added some formatting to show the
             mean.breaks = mean(breaks),    # reader I am calculating several
             sd.breaks   = sd(breaks))      # statistics.
```

```
## # A tibble: 6 x 5
## # Groups:   wool [?]
##     wool tension     n mean.breaks sd.breaks
##   <fctr>  <fctr> <int>       <dbl>     <dbl>
## 1      A       L     9    44.55556 18.097729
## 2      A       M     9    24.00000  8.660254
## 3      A       H     9    24.55556 10.272671
## 4      B       L     9    28.22222  9.858724
## 5      B       M     9    28.77778  9.431036
## 6      B       H     9    18.77778  4.893306
```

If instead of summarizing each split, we might want to just do some calculation and the output should have the same number of rows as the input data frame. In this case I'll tell `dplyr` that we are mutating the data frame instead of summarizing it. For example, suppose that I want to calculate the residual value

$$e_{ijk} = y_{ijk} - \bar{y}_{ij\cdot}$$

where $\bar{y}_{ij\cdot}$ is the mean of each `wool:tension` combination.

```
warpbreaks %>%
   group_by(wool, tension) %>%                 # group by wool:tension
   mutate(resid = breaks - mean(breaks)) %>%   # mean(breaks) of the group!
   head(  )                                    # show the first couple of rows
```

```
## # A tibble: 6 x 4
## # Groups:   wool, tension [1]
##   breaks   wool tension      resid
##    <dbl> <fctr>  <fctr>      <dbl>
## 1     26      A       L -18.555556
## 2     30      A       L -14.555556
## 3     54      A       L   9.444444
## 4     25      A       L -19.555556
## 5     70      A       L  25.444444
## 6     52      A       L   7.444444
```

1.2.3 Chaining commands together

In the previous examples we have used the %>% operator to make the code more readable but to really appreciate this, we should examine the alternative.

Suppose we have the results of a small 5K race. The data given to us is in the order that the runners signed up but we want to calculate the results for each gender, calculate the placings, and the sort the data frame by gender and then place. We can think of this process as having three steps:

1. Splitting
2. Ranking
3. Re-arranging.

```
# input the initial data
race.results <- data.frame(
  name=c('Bob', 'Jeff', 'Rachel', 'Bonnie', 'Derek', 'April','Elise','David'),
  time=c(21.23, 19.51, 19.82, 23.45, 20.23, 24.22, 28.83, 15.73),
  gender=c('M','M','F','F','M','F','F','M'))
```

We could run all the commands together using the following code:

```
arrange(
  mutate(
    group_by(
      race.results,          # using race.results
      gender),               # group by gender
    place = rank( time )),   # mutate to calculate the place column
  gender, place)             # arrange the result by gender and place
```

```
## # A tibble: 8 x 4
## # Groups:   gender [2]
##     name  time gender place
##   <fctr> <dbl> <fctr> <dbl>
## 1 Rachel 19.82      F     1
## 2 Bonnie 23.45      F     2
## 3  April 24.22      F     3
## 4  Elise 28.83      F     4
## 5  David 15.73      M     1
## 6   Jeff 19.51      M     2
## 7  Derek 20.23      M     3
## 8    Bob 21.23      M     4
```

This is very difficult to read because you have to read the code *from the inside out.*

Another (and slightly more readable) way to complete our task is to save each intermediate step of our process and then use that in the next step:

```
temp.df0 <- race.results %>% group_by( gender)
temp.df1 <- temp.df0 %>% mutate( place = rank(time) )
temp.df2 <- temp.df1 %>% arrange( gender, place )
```

It would be nice if I didn't have to save all these intermediate results because keeping track of temp1 and temp2 gets pretty annoying if I keep changing the order of how things or calculated or add/subtract steps. This is exactly what %>% does for me.

```
race.results %>%
  group_by( gender ) %>%
  mutate( place = rank(time)) %>%
  arrange( gender, place )
```

```
## # A tibble: 8 x 4
## # Groups:   gender [2]
##     name  time gender place
##   <fctr> <dbl> <fctr> <dbl>
## 1 Rachel 19.82      F     1
## 2 Bonnie 23.45      F     2
## 3  April 24.22      F     3
## 4  Elise 28.83      F     4
## 5  David 15.73      M     1
## 6   Jeff 19.51      M     2
## 7  Derek 20.23      M     3
## 8    Bob 21.23      M     4
```

1.3 Reshaping data

```
library(tidyr)   # for the gather/spread commands
library(dplyr)   # for the join stuff
```

Most of the time, our data is in the form of a data frame and we are interested in exploring the relationships. However most procedures in R expect the data to show up in a 'long' format where each row is an observation and each column is a covariate. In practice, the data is often not stored like that and the data comes to us with repeated observations included on a single row. This is often done as a memory saving technique or because there is some structure in the data that makes the 'wide' format attractive. As a result, we need a way to convert data from 'wide' to 'long' and vice-versa.

1.3.1 `tidyr`

There is a common issue with obtaining data with many columns that you wish were organized as rows. For example, I might have data in a grade book that has several homework scores and I'd like to produce a nice graph that has assignment number on the x-axis and score on the y-axis. Unfortunately this is incredibly hard to do when the data is arranged in the following way:

```
grade.book <- rbind(
  data.frame(name='Alison',  HW.1=8, HW.2=5, HW.3=8),
  data.frame(name='Brandon', HW.1=5, HW.2=3, HW.3=6),
  data.frame(name='Charles', HW.1=9, HW.2=7, HW.3=9))
grade.book
```

```
##      name HW.1 HW.2 HW.3
## 1  Alison    8    5    8
## 2 Brandon    5    3    6
## 3 Charles    9    7    9
```

What we want to do is turn this data frame from a *wide* data frame into a *long* data frame. In MS Excel this is called pivoting. Essentially I'd like to create a data frame with three columns: `name`, `assignment`, and `score`. That is to say that each homework datum really has three pieces of information: who it came from, which homework it was, and what the score was. It doesn't conceptually matter if I store it as 3 columns or 3 rows so long as there is a way to identify how a student scored on a particular homework. So we want to reshape the HW1 to HW3 columns into two columns (assignment and score).

This package was built by the sample people that created dplyr and ggplot2 and there is a nice introduction at: [http://blog.rstudio.org/2014/07/22/introducing-tidyr/]

1.3.1.1 Verbs

As with the dplyr package, there are two main verbs to remember:

1. `gather` - Gather multiple columns that are related into two columns that contain the original column name and the value. For example for columns HW1, HW2, HW3 we would gather them into two column HomeworkNumber and Score. In this case, we refer to HomeworkNumber as the key column and Score as the value column. So for any key:value pair you know everything you need.
2. `spread` - This is the opposite of gather. This takes a key column (or columns) and a results column and forms a new column for each level of the key column(s).

```
# first we gather the score columns into columns we'll name Assesment and Score
tidy.scores <- grade.book %>%
  gather( key=Assessement,  # What should I call the key column
          value=Score,      # What should I call the values column
          HW.1:HW.3         # which columns to apply this to
          )
tidy.scores
```

```
##      name Assessement Score
## 1  Alison        HW.1     8
## 2 Brandon        HW.1     5
## 3 Charles        HW.1     9
## 4  Alison        HW.2     5
## 5 Brandon        HW.2     3
## 6 Charles        HW.2     7
## 7  Alison        HW.3     8
## 8 Brandon        HW.3     6
## 9 Charles        HW.3     9
```

To spread the key:value pairs out into a matrix, we use the `spread()` command.

```
# Turn the Assessment/Score pair of columns into one column per factor level of Assessment
tidy.scores %>% spread( key=Assessement, value=Score )
```

```
##      name HW.1 HW.2 HW.3
## 1  Alison    8    5    8
## 2 Brandon    5    3    6
## 3 Charles    9    7    9
```

One way to keep straight which is the `key` column is that the key is the category, while `value` is the numerical value or response.

1.4 Storing Data in Multiple Tables

In many datasets it is common to store data across multiple tables, usually with the goal of minimizing memory used as well as providing minimal duplication of information so any change that must be made is only made in a single place.

To see the rational why we might do this, consider building a data set of blood donations by a variety of donors across several years. For each blood donation, we will perform some assay and measure certain qualities about the blood and the patients health at the donation.

```
##   Donor Hemoglobin Systolic Diastolic
## 1 Derek       17.4      121        80
## 2  Jeff       16.9      145       101
```

But now we have to ask, what happens when we have a donor that has given blood multiple times? In this case we should just have multiple rows per person along with a date column to uniquely identify a particular donation.

```
donations
```

```
##   Donor       Date Hemoglobin Systolic Diastolic
## 1 Derek 2017-04-14       17.4      120        79
## 2 Derek 2017-06-20       16.5      121        80
## 3  Jeff 2017-08-14       16.9      145       101
```

I would like to include additional information about the donor where that infomation doesn't change overtime. For example we might want to have information about the donar's birthdate, sex, blood type. However, I don't want that information in *every single donation line*. Otherwise if I mistype a birthday and have to correct it, I would have to correct it *everywhere*. For information about the donor, should live in a `donors` table, while information about a particular donation should live in the `donations` table.

Furthermore, there are many Jeffs and Dereks in the world and to maintain a unique identifier (without using Social Security numbers) I will just create a `Donor_ID` code that will uniquely identify a person. Similarly I will create a `Donation_ID` that will uniquely identify a dontation.

```
donors
```

```
##   Donor_ID F_Name L_Name B_Type      Birth       Street      City State
## 1  Donor_1  Derek    Lee     O+ 1976-09-17 7392 Willard Flagstaff    AZ
## 2  Donor_2   Jeff  Smith      A 1974-06-23     873 Vine   Bozeman    MT
```

```
donations
```

```
##   Donation_ID Donor_ID       Date Hemoglobin Systolic Diastolic
## 1  Donation_1  Donor_1 2017-04-14       17.4      120        79
## 2  Donation_2  Donor_1 2017-06-20       16.5      121        80
## 3  Donation_3  Donor_2 2017-08-14       16.9      145       101
```

If we have a new donor walk in and give blood, then we'll have to create a new entry in the `donors` table as well as a new entry in the `donations` table. If an experienced donor gives again, we just have to create a new entry in the donations table.

```
donors
```

```
##   Donor_ID F_Name L_Name B_Type      Birth       Street      City State
## 1  Donor_1  Derek    Lee     O+ 1976-09-17 7392 Willard Flagstaff    AZ
## 2  Donor_2   Jeff  Smith      A 1974-06-23     873 Vine   Bozeman    MT
## 3  Donor_3 Aubrey    Lee     O+ 1980-12-15 7392 Willard Flagstaff    AZ
```

```
donations
```

```
##   Donation_ID Donor_ID       Date Hemoglobin Systolic Diastolic
## 1  Donation_1  Donor_1 2017-04-14       17.4      120        79
## 2  Donation_2  Donor_1 2017-06-20       16.5      121        80
## 3  Donation_3  Donor_2 2017-08-14       16.9      145       101
## 4  Donation_4  Donor_1 2017-08-26       17.6      120        79
## 5  Donation_5  Donor_4 2017-08-26       16.1      137        90
```

This data storage set-up might be flexible enough for us. However what happens if somebody moves? If we don't want to keep the historical information, then we could just change the person's `Street_Address`, `City`, and `State` values. If we do want to keep that, then we could create `donor_addresses` table that contains a `Start_Date` and `End_Date` that denotes the period of time that the address was valid.

```
donor_addresses
```

```
##   Donor_ID       Street      City State Start_Date   End_Date
```

```
## 1  Donor_1 346 Treeline   Pullman    WA 2015-01-26 2016-06-27
## 2  Donor_1     645 Main Flagstsff    AZ 2016-06-28 2017-07-02
## 3  Donor_1 7392 Willard Flagstaff    AZ 2017-07-03       <NA>
## 4  Donor_2     873 Vine   Bozeman    MT 2015-03-17       <NA>
## 5  Donor_3 7392 Willard Flagstaff    AZ 2017-06-01       <NA>
```

Given this data structure, we can now easily create new donations as well as store donor information. In the event that we need to change something about a donor, there is only *one* place to make that change.

However, having data spread across multiple tables is challenging because I often want that information squished back together. For example, the blood donations services might want to find all 'O' or 'O+' donors in Flagstaff and their current mailing address and send them some notification about blood supplies being low. So we need someway to join the `donors` and `donor_addresses` tables together in a sensible manner.

1.4.1 Table Joins

Often we need to squish together two data frames but they do not have the same number of rows. Consider the case where we have a data frame of observations of fish and a separate data frame that contains information about lake (perhaps surface area, max depth, pH, etc). I want to store them as two separate tables so that when I have to record a lake level observation, I only input it *one* place. This decreases the chance that I make a copy/paste error.

To illustrate the different types of table joins, we'll consider two different tables.

```
# tibbles are just data.frames that print a bit nicer and don't automaticall
# convert character columns into factors.  They behave a bit more consistently
# in a wide variety of situations compared to data.frames.
Fish.Data <- tibble(
  Lake_ID = c('A','A','B','B','C','C'),
  Fish.Weight=rnorm(6, mean=260, sd=25) ) # make up some data
Lake.Data <- tibble(
  Lake_ID = c(    'B','C','D'),
  Lake_Name = c('Lake Elaine', 'Mormon Lake', 'Lake Mary'),
  pH=c(6.5, 6.3, 6.1),
  area = c(40, 210, 240),
  avg_depth = c(8, 10, 38))
```

```
Fish.Data
```

```
## # A tibble: 6 x 2
##   Lake_ID Fish.Weight
##     <chr>       <dbl>
## 1       A    236.5489
## 2       A    266.3815
## 3       B    235.3035
## 4       B    245.2823
## 5       C    268.3987
## 6       C    204.7937
```

```
Lake.Data
```

```
## # A tibble: 3 x 5
##   Lake_ID   Lake_Name    pH  area avg_depth
##     <chr>       <chr> <dbl> <dbl>     <dbl>
## 1       B Lake Elaine   6.5    40         8
## 2       C Mormon Lake   6.3   210        10
## 3       D   Lake Mary   6.1   240        38
```

Notice that each of these tables has a column labled `Lake_ID`. When we join these two tables, the row that describes lake `A` should be duplicated for each row in the `Fish.Data` that corresponds with fish caught from lake `A`.

```
full_join(Fish.Data, Lake.Data)
```

```
## Joining, by = "Lake_ID"
```

```
## # A tibble: 7 x 6
##   Lake_ID Fish.Weight   Lake_Name    pH  area avg_depth
##     <chr>       <dbl>       <chr> <dbl> <dbl>     <dbl>
## 1       A    236.5489        <NA>    NA    NA        NA
## 2       A    266.3815        <NA>    NA    NA        NA
## 3       B    235.3035 Lake Elaine   6.5    40         8
## 4       B    245.2823 Lake Elaine   6.5    40         8
## 5       C    268.3987 Mormon Lake   6.3   210        10
## 6       C    204.7937 Mormon Lake   6.3   210        10
## 7       D          NA   Lake Mary   6.1   240        38
```

Notice that because we didn't have any fish caught in lake `D` and we don't have any Lake information about lake `A`, when we join these two tables, we end up introducing missing observations into the resulting data frame.

The other types of joins govern the behavor or these missing data.

`left_join(A, B)` For each row in A, match with a row in B, but don't create any more rows than what was already in A.

`inner_join(A,B)` Only match row values where both data frames have a value.

```
left_join(Fish.Data, Lake.Data)
```

```
## Joining, by = "Lake_ID"
```

```
## # A tibble: 6 x 6
##   Lake_ID Fish.Weight   Lake_Name    pH  area avg_depth
##     <chr>       <dbl>       <chr> <dbl> <dbl>     <dbl>
## 1       A    236.5489        <NA>    NA    NA        NA
## 2       A    266.3815        <NA>    NA    NA        NA
## 3       B    235.3035 Lake Elaine   6.5    40         8
## 4       B    245.2823 Lake Elaine   6.5    40         8
## 5       C    268.3987 Mormon Lake   6.3   210        10
## 6       C    204.7937 Mormon Lake   6.3   210        10
```

```
inner_join(Fish.Data, Lake.Data)
```

```
## Joining, by = "Lake_ID"
```

```
## # A tibble: 4 x 6
##   Lake_ID Fish.Weight   Lake_Name    pH  area avg_depth
##     <chr>       <dbl>       <chr> <dbl> <dbl>     <dbl>
## 1       B    235.3035 Lake Elaine   6.5    40         8
## 2       B    245.2823 Lake Elaine   6.5    40         8
## 3       C    268.3987 Mormon Lake   6.3   210        10
## 4       C    204.7937 Mormon Lake   6.3   210        10
```

The above examples assumed that the column used to join the two tables was named the same in both tables. This is good practice to try to do, but sometimes you have to work with data where that isn't the case. In that situation you can use the `by=c("ColName.A"="ColName.B")` syntax where `ColName.A` represents the name of the column in the first data frame and `ColName.B` is the equivalent column in the second data frame.

1.5 Exercises

1. The dataset `ChickWeight` tracks the weights of 48 baby chickens (chicks) feed four different diets.

   a. Load the dataset using

   ```
   data(ChickWeight)
   ```

   b. Look at the help files for the description of the columns.

   c) Remove all the observations except for the weights on day 10 and day 20.

   d) Calculate the mean and standard deviation for each diet group on days 10 and 20.

2. The OpenIntro textbook on statistics includes a data set on body dimensions.

   a) Load the file using

   ```
   Body <- read.csv('http://www.openintro.org/stat/data/bdims.csv')
   ```

   b) The column sex is coded as a 1 if the individual is male and 0 if female. This is a non-intuitive labeling system. Create a new column `sex.MF` that uses labels Male and Female.

   c) The columns wgt and hgt measure weight and height in kilograms and centimeters (respectively). Use these to calculate the Body Mass Index (BMI) for each individual where

   $$BMI = \frac{Weight\,(kg)}{[Height\,(m)]^2}$$

   d) Double check that your calculated BMI column is correct by examining the summary statistics of the column. BMI values should be between 18 to 40 or so. Did you make an error in your calculation?

   e) The function `cut` takes a vector of continuous numerical data and creates a factor based on your give cut-points.

   ```
   # Define a continuous vector to convert to a factor
   x <- 1:10

   # divide range of x into three groups of equal length
   cut(x, breaks=3)

   # divide x into four groups, where I specify all 5 break points
   cut(x, breaks = c(0, 2.5, 5.0, 7.5, 10))
   # (0,2.5] (2.5,5] means 2.5 is included in first group
   # right=FALSE changes this to make 2.5 included in the second

   # divide x into 3 groups, but give them a nicer
   # set of group names
   cut(x, breaks=3, labels=c('Low','Medium','High'))
   ```

   Create a new column of in the data frame that divides the age into decades (10-19, 20-29, 30-39, etc). Notice the oldest person in the study is 67.

   ```
   Body <- Body %>%
     mutate( Age.Grp = cut(age,
                           breaks=c(10,20,30,40,50,60,70),
                           right=FALSE))
   ```

   f) Find the average BMI for each Sex and Age group.

3. Suppose we are given information about the maximum daily temperature from a weather station in Flagstaff, AZ. The file is available at the GitHub site that this book is hosted on.

```
FlagTemp <-  read.csv(
  'https://github.com/dereksonderegger/570L/raw/master/data-raw/FlagMaxTemp.csv',
   header=TRUE, sep=',')
```

   This file is in a wide format, where each row represents a month and the columns X1, X2, ..., X31 represent the day of the month the observation was made.

   a. Convert data set to the long format where the data has only four columns: `Year`, `Month`, `Day`, `Tmax`.

   b. Calculate the average monthly maximum temperature for each Month in the dataset (So there will be 365 mean maximum temperatures). *You'll probably have some issues taking the mean because there are a number of values that are missing and by default R refuses to take means and sums when there is missing data. The argument* `na.rm=TRUE` *to* `mean()` *allows you to force R to remove the missing observations before calculating the mean.*

   c. Convert the average month maximums back to a wide data format where each line represents a year and there are 12 columns of temperature data (one for each month) along with a column for the year. *There will be a couple of months that still have missing data because the weather station was out of commision for those months and there was NO data for the entire month.*

4. A common task is to take a set of data that has multiple categorical variables and create a table of the number of cases for each combination. An introductory statistics textbook contains a dataset summarizing student surveys from several sections of an intro class. The two variables of interest for us are `Gender` and `Year` which are the students gender and year in college.

   a. Download the dataset and correctly order the `Year` variable using the following:

```
Survey <- read.csv('http://www.lock5stat.com/datasets/StudentSurvey.csv', na.strings=c('',' '))
   mutate(Year = factor(Year, levels=c('FirstYear','Sophomore','Junior','Senior')))
```

   b. Using some combination of `dplyr` functions, produce a data set with eight rows that contains the number of responses for each gender:year combination. *Notice there are two females that neglected to give their Year and you should remove them first. The function* `is.na(Year)` *will return logical values indicating if the Year value was missing and you can flip those values using the negation operator* `!`*. So you might consider using* `!is.na(Year)` *as the argument to a filter() command. Alternatively you sort on Year and remove the first two rows using* `slice(-2:-1)`*. Next you'll want to summarize each Year/Gender group using the* `n()` *function which gives the number of rows in a data set.*

   c. Using `tidyr` commands, produce a table of the number of responses in the following form:

Gender	First Year	Sophmore	Junior	Senior
Female				
Male				

5. The package `nycflights` contains information about all the flights that arrived in or left from New York City in 2013. This package contains five data tables, but there are three data tables we will work with. The data table `flights` gives information about a particular flight, `airports` gives information about a particular airport, and `airlines` gives information about each airline. Create a table of all the flights on February 14th by Virgin America that has columns for the carrier, destination, departure time, and flight duration. Join this table with the airports information for the destination. Notice that because the column for the destination airport code doesn't match up between `flights` and `airports`, you'll have to use the `by=c("TableA.Col"="TableB.Col")` argument where you insert the correct

names for `TableA.Col` and `TableB.Col`.

Chapter 2

Markov Chain Monte Carlo

```
library(ggplot2)
library(dplyr)
library(devtools)
install_github('dereksonderegger/STA578') # Some routines I created for this
library(STA578)
```

Modern statistical methods often rely on being able to produce random numbers from an arbitrary distribution. In this chapter we will explore how this is done.

2.1 Generating $U \sim Uniform(0,1)$

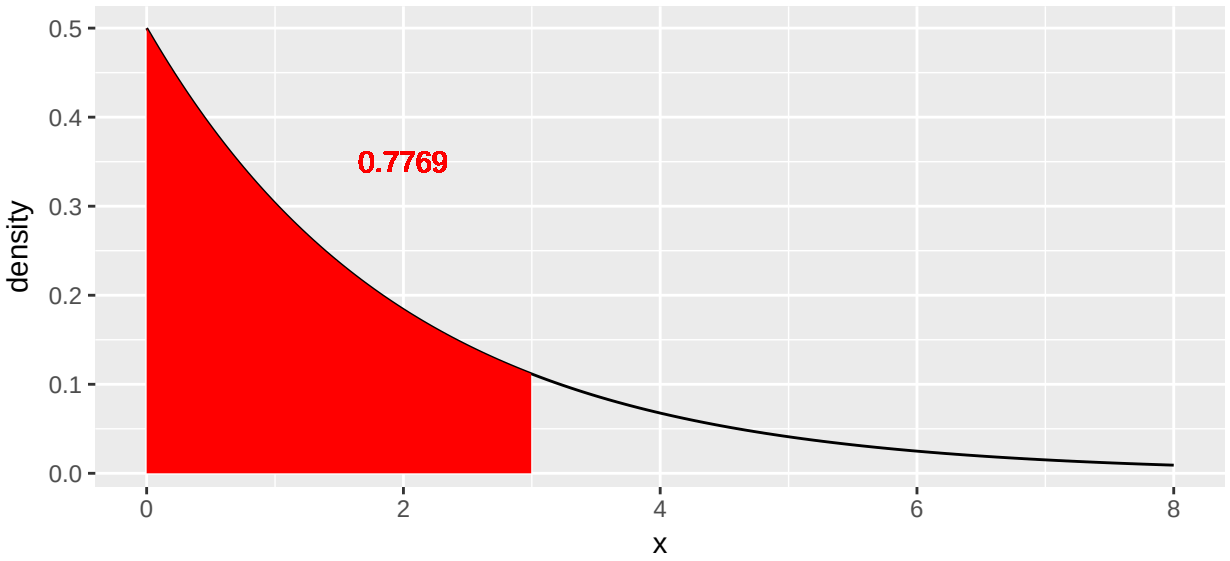
It is extremely difficult to generate actual random numbers but it is possible to generate pseudo-random numbers. That is, we'll generate a sequence of digits, say 1,000,000,000 long such that any sub-sequence looks like we are just drawing digits [0-9] randomly. Then given this sequence, we chose a starting point (perhaps something based off the clock time of the computer we are using). From the starting point, we generate $U \sim Uniform(0,1)$ numbers by using just reading off successive digits.

In practice there are many details of the above algorithm that are quite tricky, but we will ignore those issues and assume we have some method for producing random samples from $Uniform\,(0,1)$ distribution.

2.2 Inverse CDF Method

Suppose that we wish to generate a random sample from a given distribution, say, $X \sim Exp\,(\lambda = 1/2)$. This distribution is pretty well understood and it is easy to calculate various probabilities. The density function is $f\,(x) = \lambda \cdot e^{-x\lambda}$ and we can easily calculate probabilities such as

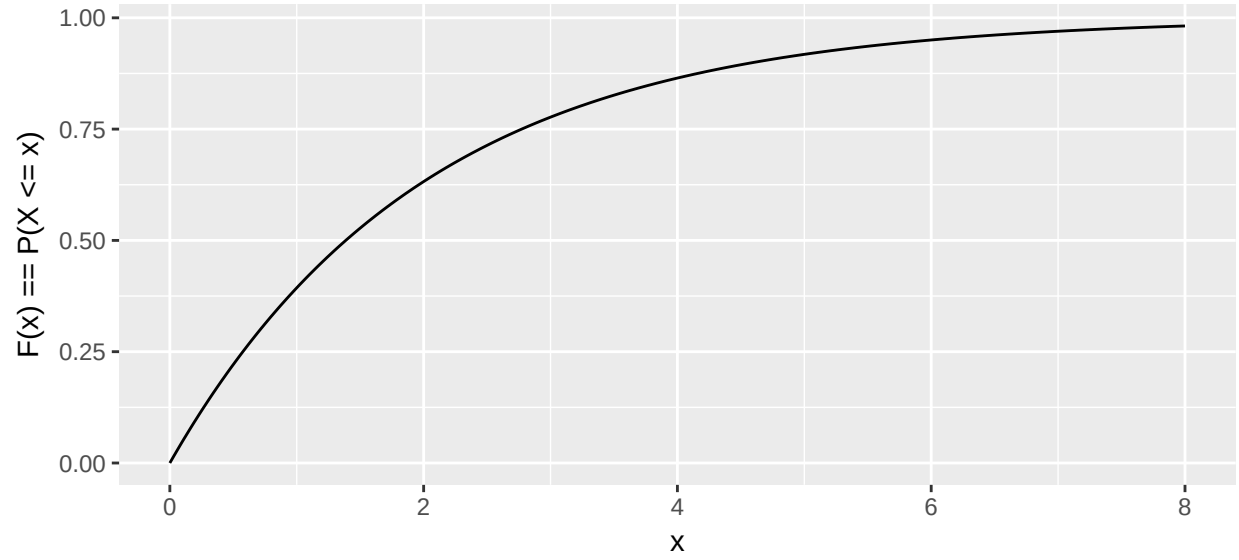
$$\begin{aligned} P\,(X \leq 3) &= \int_0^3 \lambda e^{-x\lambda}\,dx \\ &= e^{-0\lambda} - e^{-3\lambda} \\ &= 1 - e^{-3\lambda} \\ &= 0.7769 \end{aligned}$$



Given this result, it is possible to figure out $P(X \leq x)$ for any value of x. (Here the capital X represents the random variable and the lower case x represents a particular value that this variable can take on.) Now thinking of these probabilities as a function, we define the cumulative distribution function (CDF) as

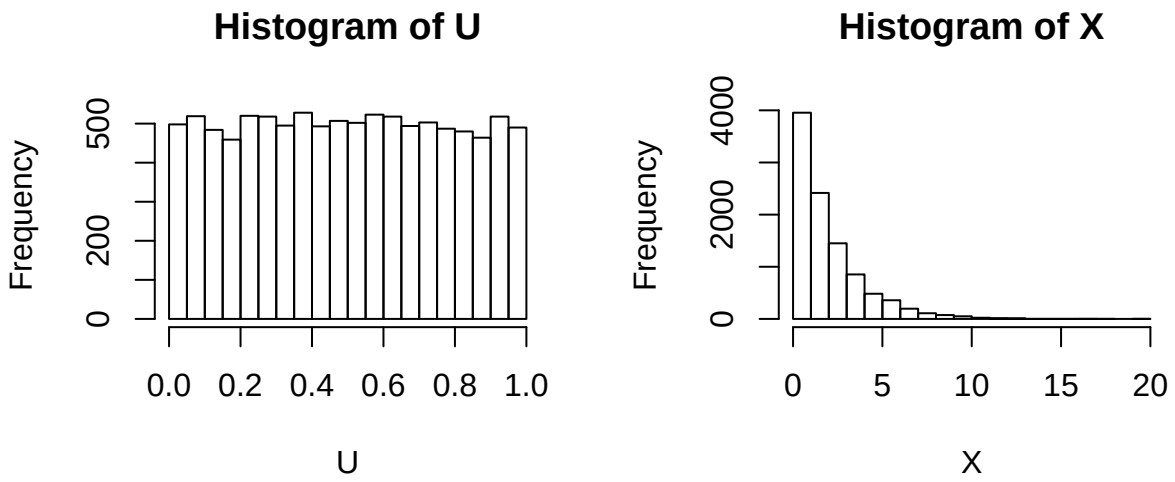
$$F(x) = P(X \leq x) = 1 - e^{-x\lambda}$$

If we make a graph of this function we have



Given this CDF, we if we can generate a $U \sim Uniform(0, 1)$, we can just use the CDF function in reverse (i.e the inverse CDF) and transform the U to be an $X \sim Exp(\lambda)$ random variable. In R, most of the common distributions have a function that calculates the inverse CDF, in the exponential distribution it is `qexp(x, rate)` and for the normal it would be `qnorm()`, etc.

```
U <- runif(10000, min=0, max=1)  # 10,000 Uniform(0,1) values
X <- qexp(U, rate=1/2)
par(mfrow=c(1,2))
hist(U)
hist(X)
```

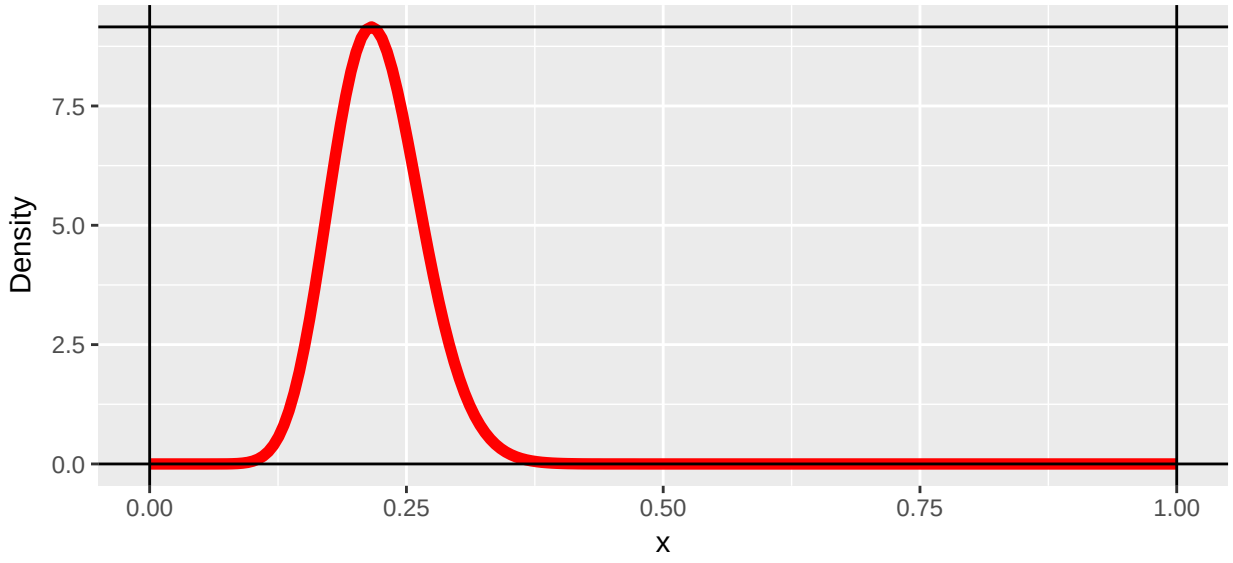


This is the type of trick that Statistics students might learn in a probability course, but this is hardly interesting from a computationally intensive perspective, so if you didn't follow the calculus, don't fret.

2.3 Accept/Reject Algorithm

We now consider a case where we don't know the CDF (or it is really hard to work with). Let the random variable X which can take on values from $0 \leq X \leq 1$ and has probability density function $f(x)$. Furthermore, suppose that we know what the maximum value of the density function is, which we'll denote $M = \max f(x)$.

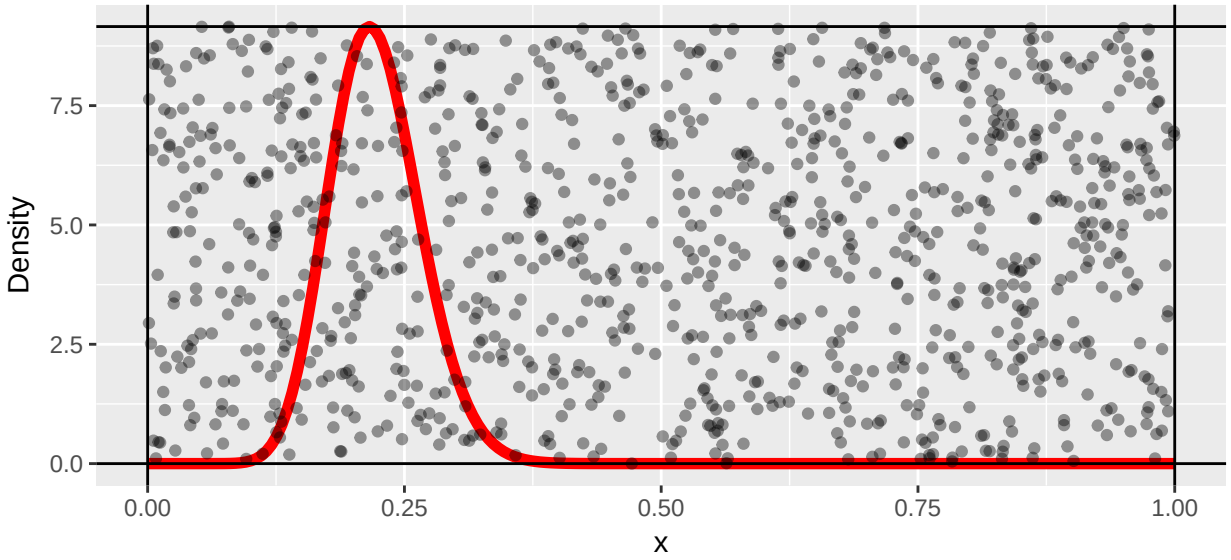
```
x <- seq(0,1, length=200)
y <- dbeta(x, 20, 70)
M <- max(y)
data.line <- data.frame(x=x, y=y)
p <- ggplot(data.line, aes(x=x, y=y)) + geom_line(color='red', size=2) +
  labs(y='Density') +
  geom_vline(xintercept=c(0,1)) +
  geom_hline(yintercept=c(0, max(y)))
p
```



It is trivial to generate points that are uniformly distributed in the rectangle by randomly selecting points

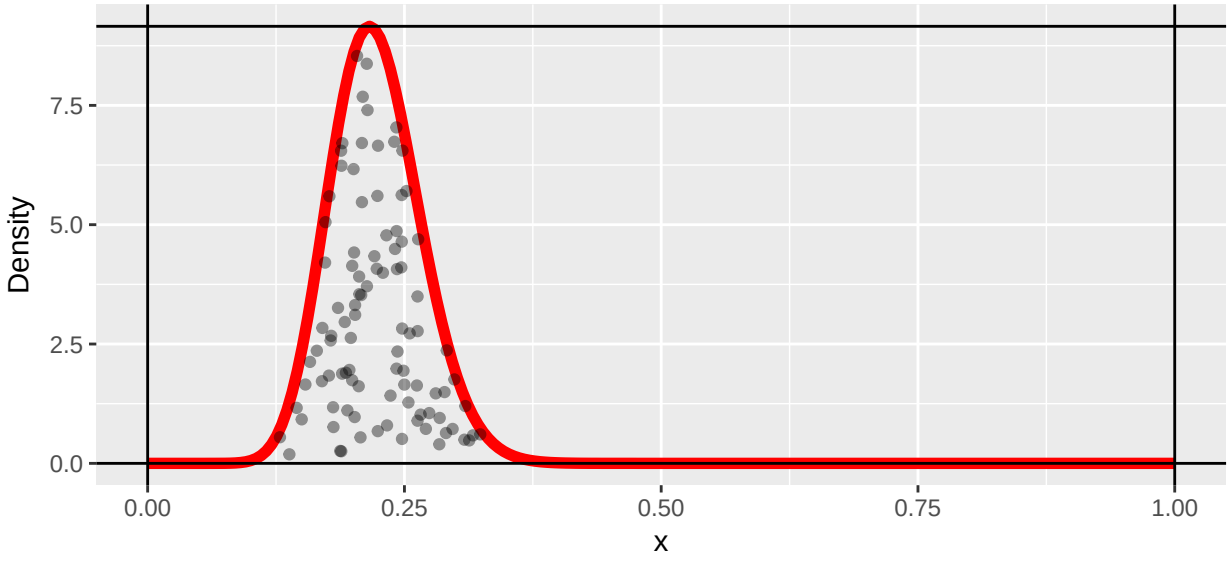
(x_i, y_i) by letting x_i be randomly sampled from a $Uniform(0,1)$ distribution and y_i be sampled from a $Uniform(0, M)$ distribution. Below we sample a thousand points.

```
N <- 1000
x <- runif(N, 0,1)
y <- runif(N, 0, M)
proposed <- data.frame(x=x, y=y)
p + geom_point(data=proposed, alpha=.4)
```



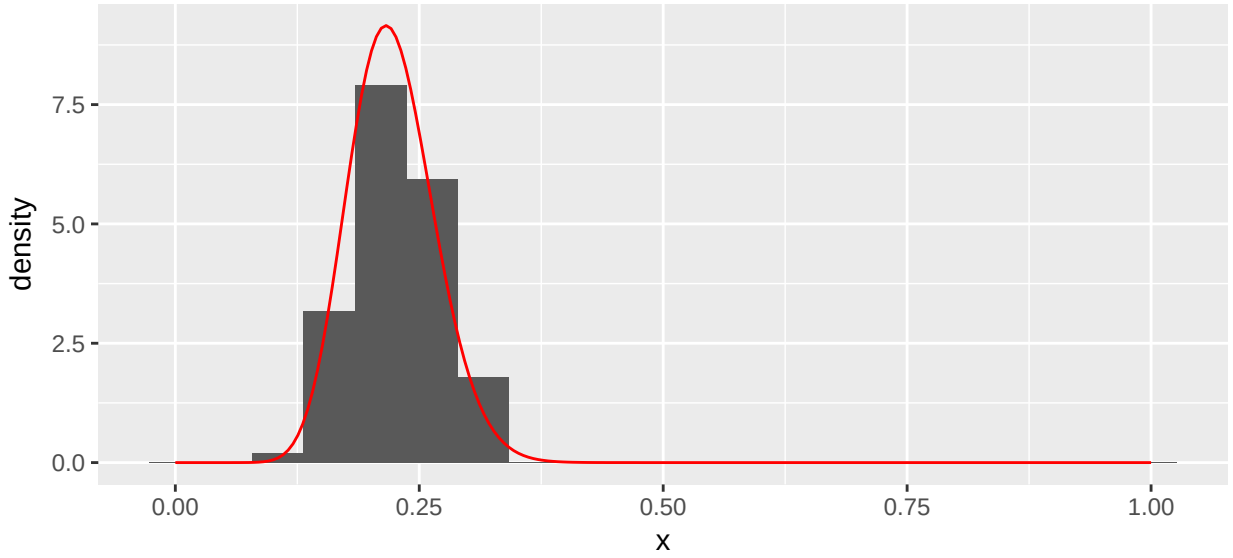
Since we want to select a random sample from the curved distribution (and not uniformly from the box), I will reject pairs (x_i, y_i) if $y_i \geq f(x_i)$. This leaves us with the following points.

```
accepted <- proposed %>%
  filter( y <= dbeta(x, 20,70) )
p + geom_point(data=accepted, alpha=.4)
```



We can now regard those x_i values as a random sample from the distribution $f(x)$ and the histogram of those values is a good approximation to the distribution $f(x)$.

```
ggplot(data=accepted, aes(x=x)) +
  geom_histogram(aes(y=..density..), bins=20) +
  geom_line(data=data.line, aes(x=x,y=y), color='red')
```



Clearly this is a fairly inefficient way to generate points from a distribution, but it contains a key concept

$$x_i \text{ is accepted if } u_i < \frac{f(x_i)}{M}$$

where $y_i = u_i \cdot M$ is the height of the point and u_i is a random observation from the $Uniform(0,1)$ distribution.

Pros/Cons

- Pro: This technique is very simple to program.
- Cons: Need to know the maximum height of the distribution.
- Cons: This can be a very inefficient algorithm as most of the proposed values are thrown away.

2.4 MCMC algorithm

Suppose that we have one observation, x_1, from the distribution $f(x)$ and we wish to create a whole sequence of observations $x_2, \ldots, x_n$ that are also from that distribution. The following algorithm will generate x_{i+1} given the value of x_i.

1. Generate a proposed x^*_{i+1} from a distribution that is symmetric about x_i. For example $x^*_{i+1} \sim N(x_i, \sigma = 1)$.
2. Generate a random variable u_{i+1} from a $Uniform(0,1)$ distribution.
3. If
$$u_{i+1} \leq \frac{f(x^*_{i+1})}{f(x_i)}$$
then we accept x^*_{i+1} and define $x_{i+1} = x^*_{i+1}$, otherwise reject x^*_{i+1} and define $x_{i+1} = x_i$

The idea of this algorithm is that if we propose an x^*_{i+1} value that has a higher density than x_i, we should always accept that value as the next observation in our chain. If we propose a value that has lower density, then we should *sometimes* accept it, and the correct probability of accepting it is the ratio of the probability

density function. If we propose a value that has a much lower density, then we should rarely accept it, but if we propose a value that is has only a slightly lower density, then we should usually accept it.

There is theory that show that a large sample drawn as described will have the desired proportions that match the distribution of interest. However, the question of "how large is large enough" is a very difficult question.

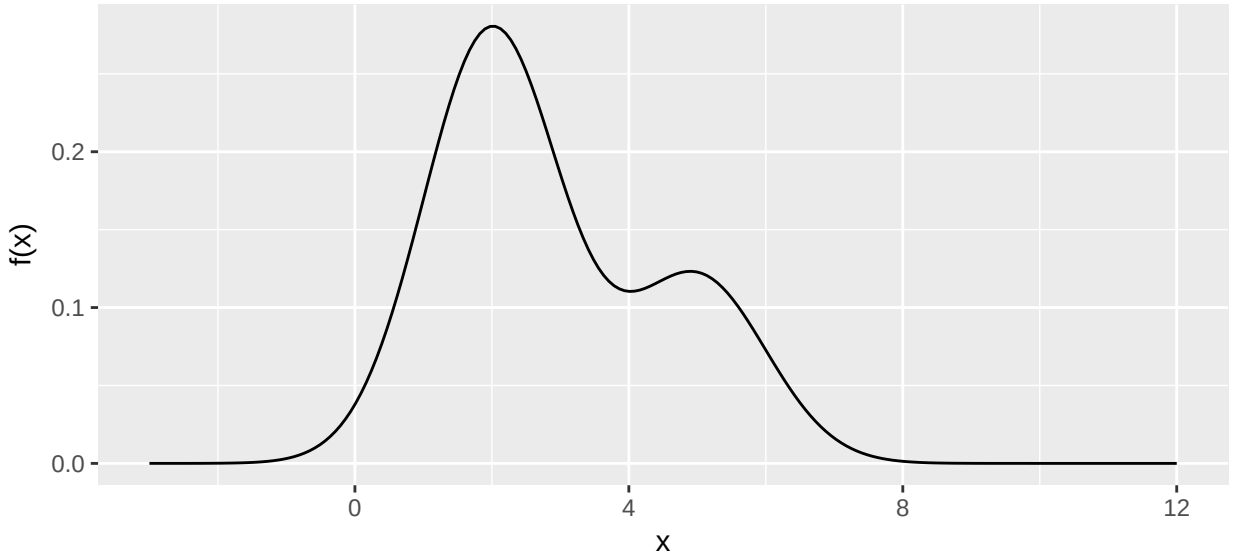
Actually we only need to know the distribution up to a scaling constant. Because we'll look at the ratio $f(x^*)/f(x)$ any constants that don't depend on x will cancel. For Bayesians, this is a crucial point.

2.4.1 Mixture of normals

We consider the problem of generating a random sample from a mixture of normal distributions. We first define our distribution $f(x)$

```
df <- function(x){
  return(.7*dnorm(x, mean=2, sd=1) + .3*dnorm(x, mean=5, sd=1))
}
```

```
x <- seq(-3,12, length=200)
density.data <- data.frame(x=x, y=df(x))
density <- ggplot( density.data, aes(x=x, y=y)) +
  geom_line() +
  labs(y='f(x)', x='x')
density
```



It is worth noting at this point that the variance of this distribution is $\sigma = 2.89$.

```
mu      <- integrate( function(x){ x * df(x) },        lower=-Inf, upper=Inf)
mu <- mu$value
sigma2 <- integrate( function(x){ (x-mu)^2 * df(x) },  lower=-Inf, upper=Inf)
c( mu=mu, sigma2=sigma2$value )
```

```
##     mu sigma2
##   2.90   2.89
```

Next we define our proposal distribution, which will be $Uniform(x_i - 2, x_i + 2)$.

```
rproposal <- function(x.i){
  out <- x.i + runif(1, -2, 2)  # x.i + 1 observation from Uniform(-2, 2)
  return(out)
}
```

Starting from $x_1 = 3$, we will randomly propose a new value.

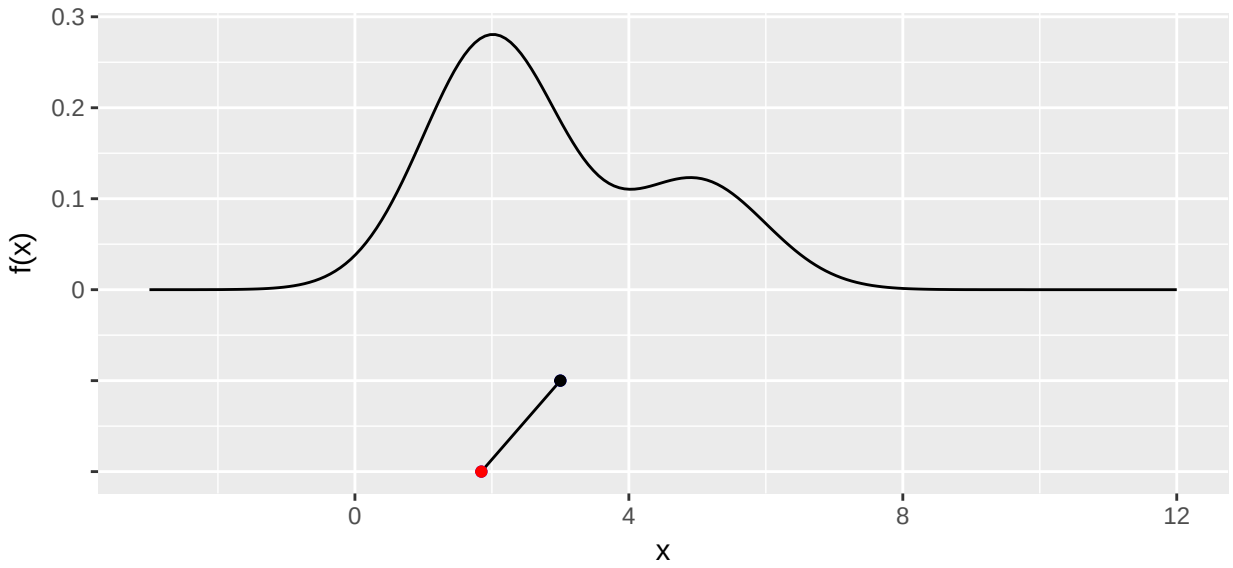
```
x <- 3;       # starting value for the chain
x.star <- 3   # initialize the vector of proposal values
x.star[2] <- rproposal( x[1] )
x.star[2]
```

```
## [1] 1.846296
```

```
if( df(x.star[2]) / df(x[1])  >  runif(1) ){
  x[2] <- x.star[2]
}else{
  x[2] <- x[1]
}
```

We proposed a value of $x = 1.846$ and because $f(1.846) \geq f(3)$ then this proposed value must be accepted (because the ratio $f(x_2^*)/f(x_1) \geq 1$ and is therefore greater than $u_2 \in [0, 1]$. We plot the chain below the density plot to help visualize the process.

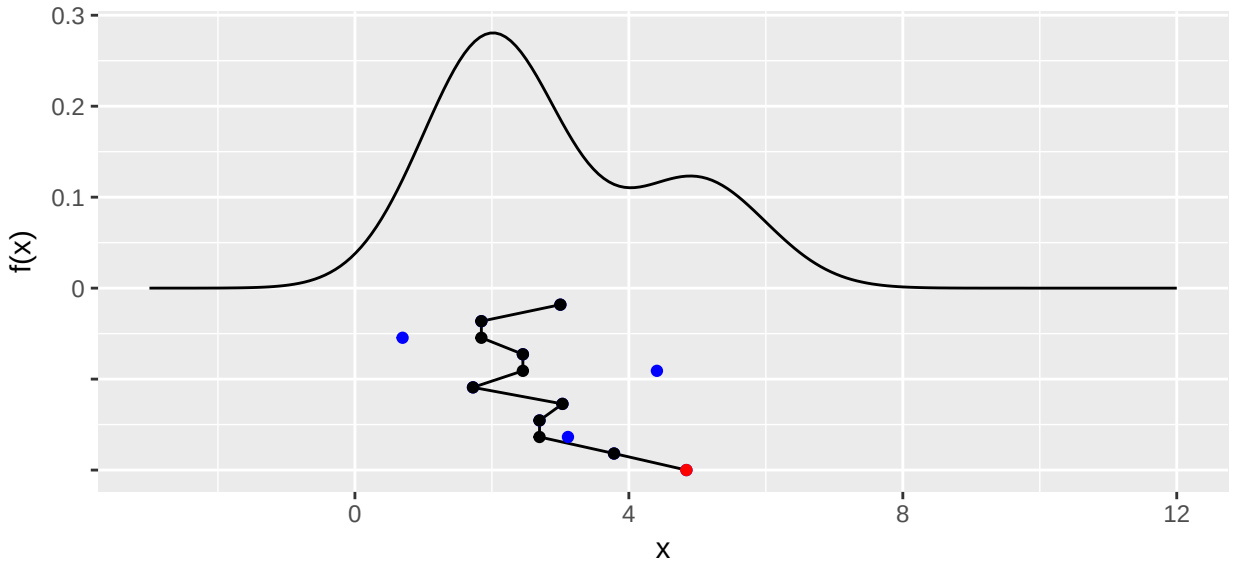
```
STA578::plot_chain(density, x, x.star)
```



Next we will repeat the process for ten iterations. If a proposed point is rejected, it will be shown in blue and not connected to the chain. The connected line will be called a traceplot and is usually seen rotated counter-clockwise by 90 degrees.

```
for( i in 2:10 ){
  x.star[i+1] <- rproposal( x[i] )
  if( df(x.star[i+1]) / df(x[i]) > runif(1) ){
    x[i+1] <- x.star[i+1]
  }else{
    x[i+1] <- x[i]
  }
}
```

```
STA578::plot_chain(density, x, x.star)
```



We see that the third step in the chain, we proposed a value near $x = 1$ and it was rejected. Similarly steps 5 and 9 were rejected because $f\left(x_{i+1}^*\right)$ of the proposed value was very small compared to $f\left(x_i\right)$ and thus the proposed value was unlikely to be accepted.

At this point, we should make one final adjustment to the algorithm. In general, it is not necessary for the proposal distribution to by symmetric about x_i. Let $g\left(x_{i+1}^*|x_i\right)$ be the density function of the proposal distribution, then the appropriate accept/reject criteria is modified to be

- If

$$u_{i+1} \leq \frac{f\left(x_{i+1}^*\right)}{f\left(x_i\right)} \frac{g\left(x_i|x_{i+1}^*\right)}{g\left(x_{i+1}^*|x_i\right)}$$

then we accept x_{i+1}^* and define $x_{i+1} = x_{i+1}^*$, otherwise reject x_{i+1}^* and define $x_{i+1} = x_i$

Notice that if $g\left(\right)$ is symmetric about x_i then $g\left(x_i|x_{i+1}^*\right) = g\left(x_{i+1}^*|x_i\right)$ and the second fraction is simply 1.

It will be convenient to hide the looping part of the MCMC, so we'll create a simple function to handle the details. We'll also make a program to make a traceplot of the chain.

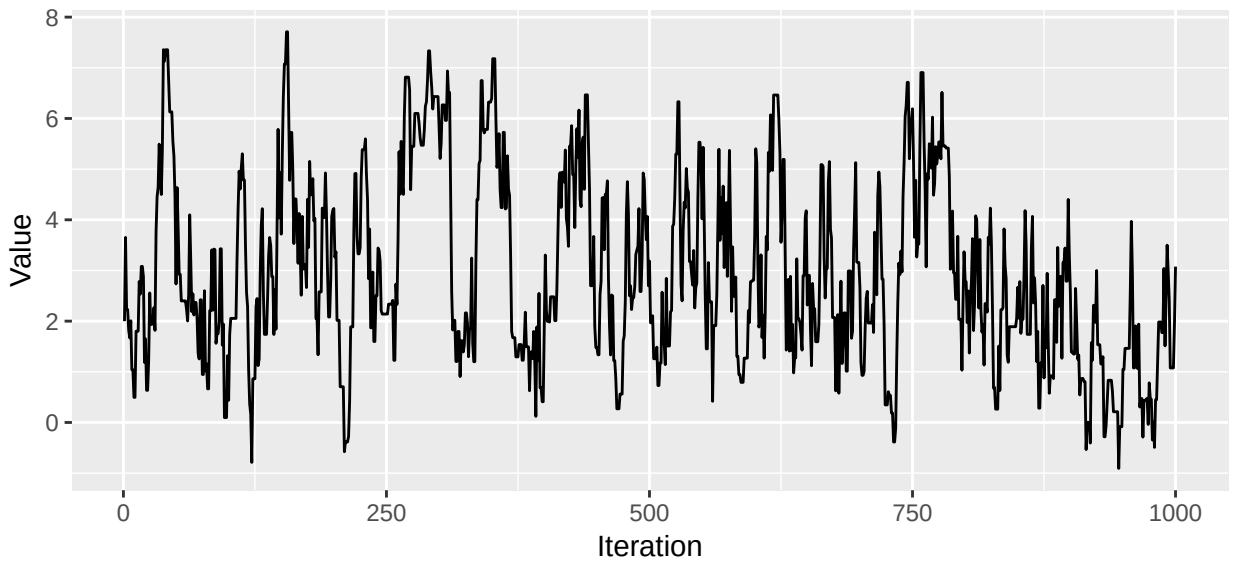
```
?MCMC
# You can look at the source code of many functions by just typing the name.
# This trick unfortunately doesn't work with functions that are just wrappers
# to C or C++ programs or are hidden due to Object Oriented inheritance.
MCMC
```

```
## function (df, start, rprop, dprop = NULL, N = 1000)
## {
##     if (is.null(dprop)) {
##         dprop <- function(to, from) {
##             1
##         }
##     }
##     chain <- rep(NA, N)
##     chain[1] <- start
##     for (i in 1:(N - 1)) {
##         x.star <- rprop(chain[i])
##         r1 <- df(x.star)/df(chain[i])
```

```
##         r2 <- dprop(chain[i], x.star)/dprop(x.star, chain[i])
##         if (r1 * r2 > runif(1)) {
##             chain[i + 1] <- x.star
##         }
##         else {
##             chain[i + 1] <- chain[i]
##         }
##     }
##     return(chain)
## }
## <environment: namespace:STA578>
```

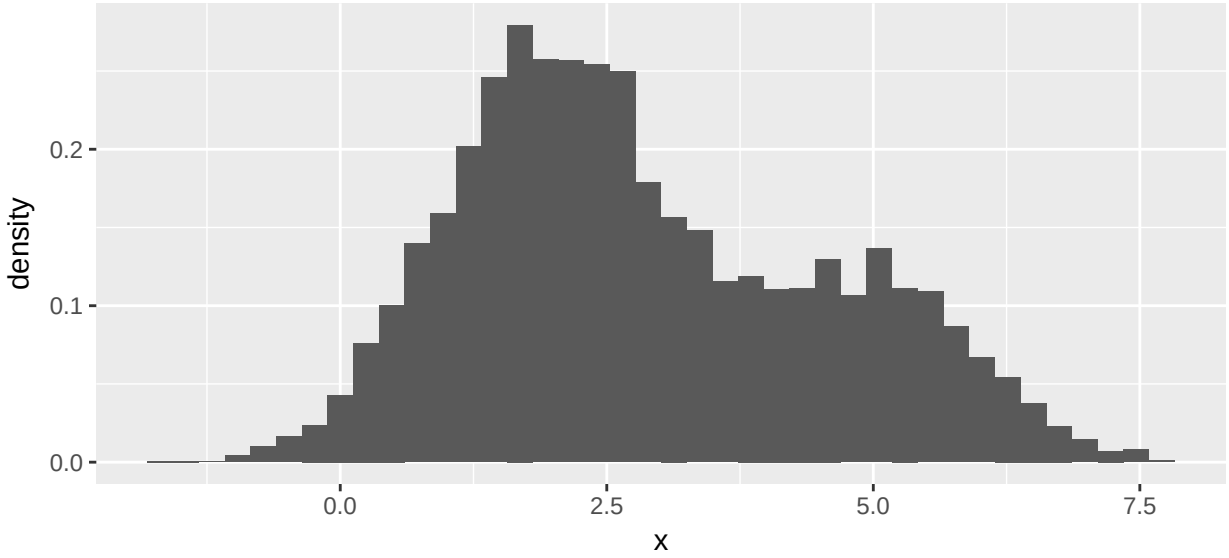
We should let this process run for a long time, and we will just look the traceplot.

```
chain <- STA578::MCMC(df, 2, rproposal, N=1000) # do this new!
STA578::trace_plot(chain)
```



This seems like it is working as the chain has occasional excursions out tho the tails of the distribution, but is mostly concentrated in the peaks of the distribution. We'll run it longer and then examine a histogram of the resulting chain.

```
chain2 <- MCMC(df, chain[1000], rproposal, N=10000)
long.chain <- c(chain, chain2)
ggplot( data.frame(x=long.chain), aes(x=x, y=..density..)) +
  geom_histogram(bins=40)
```



All in all this looks quite good.

2.4.2 Common problems

Proposal Distribution

The proposal distribution plays a critical role in how well the MCMC simulation explores the distribution of interest. If the variance of the proposal distribution is too small, then the number of steps to required to move a significant distance across the distribution becomes large. If the variance is too large, then a large percentage of the proposed values will be far from the center of the distribution and will be rejected.

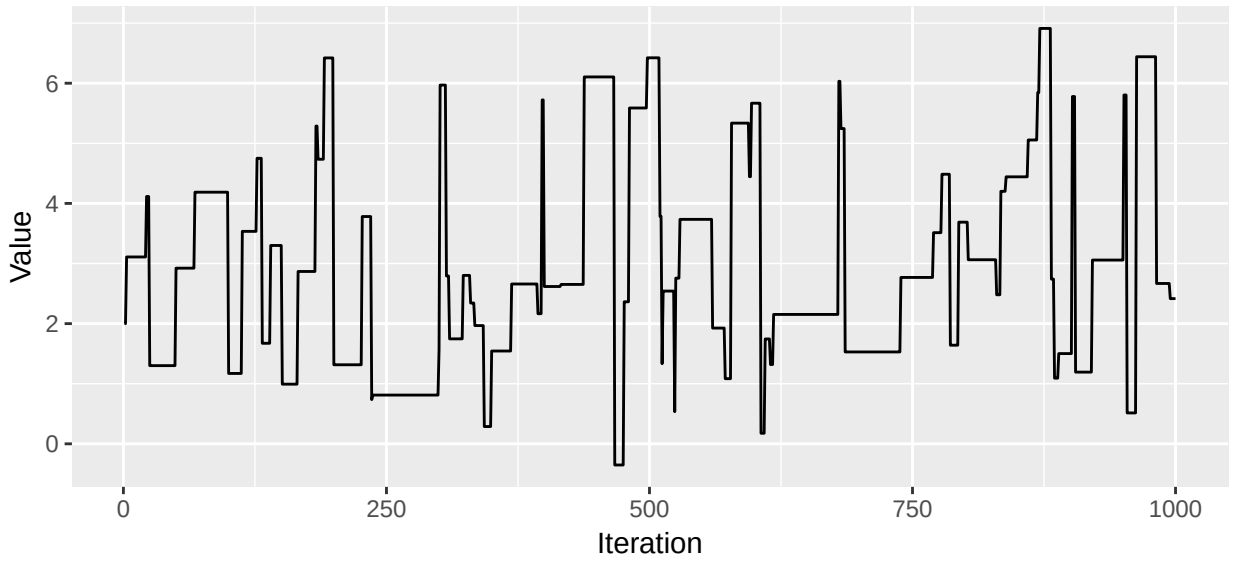
First, we create a proposal distribution with a small variance and examine its behavior.

```
p.small <- function(x){
  return( x + runif(1, -.1, +.1) )
}
chain <- MCMC(df, 2, p.small, N=1000)
trace_plot(chain)
```

In these 1000 steps, the chain has not gone smaller than 1 or larger than 3.5 and hasn't explored the second "hump" of the distribution yet. If the valley between the two humps was deeper, it is possible that the chain would never manage to cross the valley and find the second hump.

The effect of too large of variance is also troublesome.

```
p.large <- function(x){
  return( x + runif(1, -30, +30) )
}
chain <- MCMC(df, 2, p.large, N=1000)
trace_plot(chain)
```



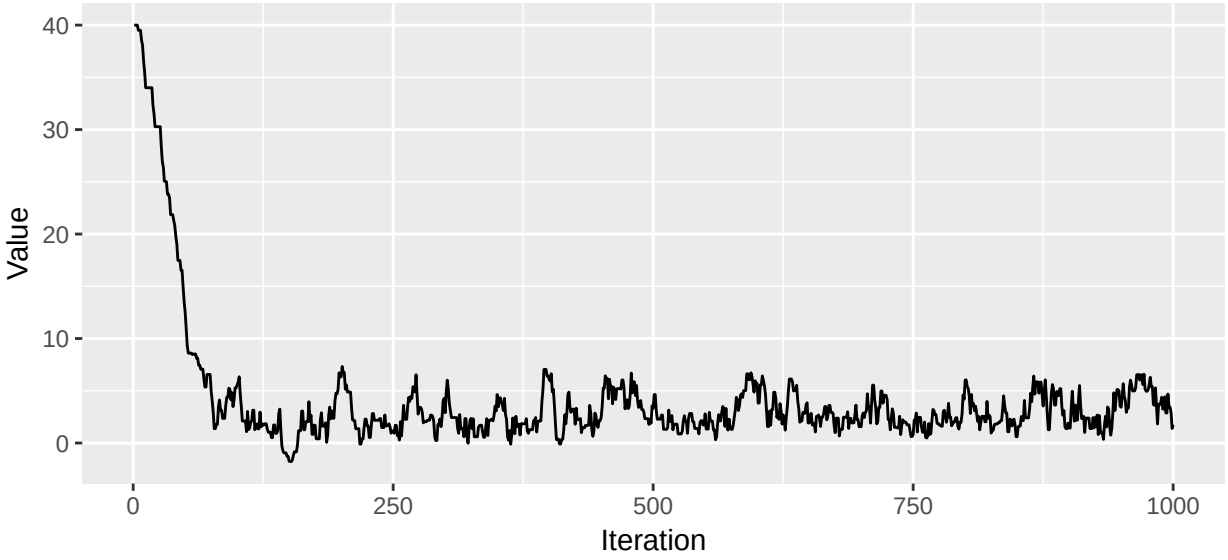
This chain is not good because it stays in place for too long before finding another suitable value. It will take a very long time to suitably explore the distribution.

Given these two examples, it is clear that the choice of the variance parameter is a critical choice. A commonly used rule of thumb that balances these two issues is that the variance parameter should be chosen such that approximately $40 - 60\%$ of the proposed values are accepted. To go about finding that variance parameter, we will have a initialization period where we tweak the variance parameter until we are satisfied with the acceptance rate.

Burn-in

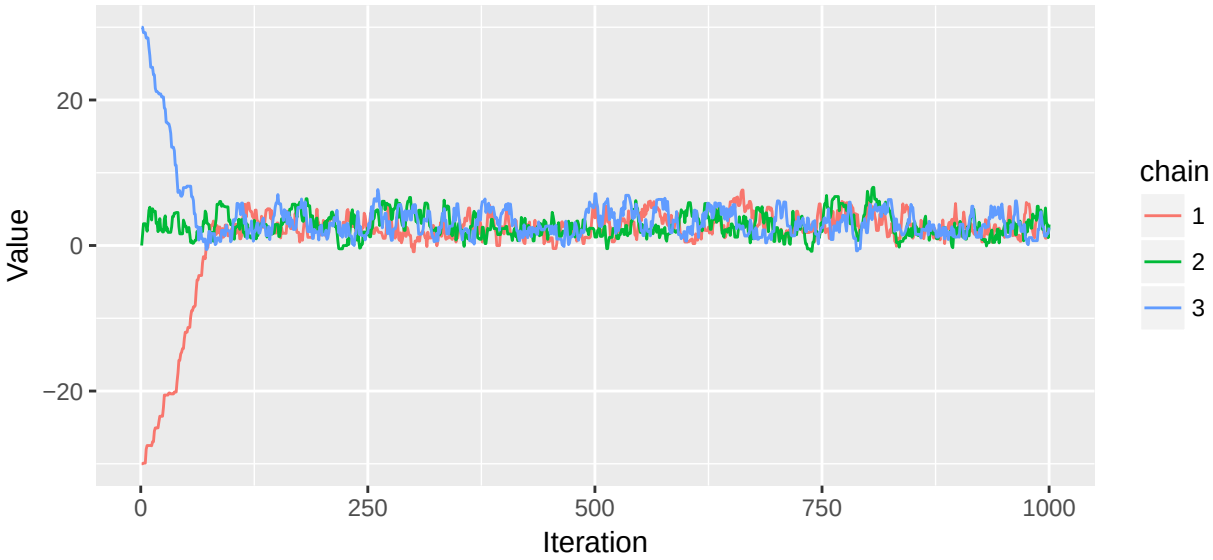
Another problem with the MCMC algorithm is that it requires that we already have a sample drawn from the distribution in question. This is clearly not possible, so what we do is use a random start and hope that the chain eventually finds the center of the distribution.

```
chain <- MCMC(df, 40, rproposal, N=1000)  # start at x = 40
trace_plot(chain)
```



Given this example, it seems reasonable to start the chain and then disregard the initial "burn-in" period. However, how can we tell the difference between a proposal distribution with too small of variance, and a starting value that is far from the center of the distribution? One solution is to create multiple chains with different starting points. When all the chains become well mixed and indistinguishable, we'll use the remaining observations in the chains as the sample.

```
chain1 <- MCMC(df, -30, rproposal, N=1000)
chain2 <- MCMC(df, 0,   rproposal, N=1000)
chain3 <- MCMC(df, 30,  rproposal, N=1000)
chains <- cbind(chain1, chain2, chain3)
trace_plot(chains)
```



2.4.3 Assessing Chain Convergence

Perhaps the most convenient way to asses convergence is just looking at the traceplots, but it is also good to derive some quantitative measurements of convergence. One idea from ANOVA is that the variance of the distribution we are sampling from, σ^2, could be estimated using the within chain variability or it could be estimated using the variance between the chains.

Let m be the number of chains and n be the number of samples per chain. We define W be the average of the within chain sample variances. Formally we let s_i^2 be the sample variance of the i^{th} chain and

$$W = \frac{1}{m}\sum_{i=1}^{m} s_i^2$$

We can interpret W as the amount of wiggle within each chain.

Next we recognize that under perfect sampling and large sample sizes, the chains should be right on top of each other, and the mean of each chain should be very very close to the mean of all the other chains. Mathematically, we'd say the mean of each chain has a distribution $\bar{x}_i \sim N\left(\mu, \frac{\sigma^2}{n}\right)$ where μ is the mean of the distribution we are approximating. Doing some probability, this implies that $n\bar{x}_i \sim N\left(n\mu, \sigma^2\right)$. Next, using the m chain means, we could estimate σ^2 by looking at the variances of the chain means! Let $\bar{x}_{\cdot\cdot}$ be the mean of all observation and $\bar{x}_{i\cdot}$ be the mean of the i^{th} chain. Let

$$B = \frac{n}{m-1}\sum_{i=1}^{m}(\bar{x}_{i\cdot} - \bar{x}_{\cdot\cdot})^2$$

Consider chains after we have removed the initial burn-in period where the chains obviously don't overlap. Notice that both W and B are estimating σ^2, but if the MCMC has not converged (i.e. the space explored by each chains doesn't perfectly overlap the space explored by the other chains), then the difference in means will be big and as a result, B will be much larger than σ^2. Likewise, W will be much smaller than σ^2 because the chains won't have fully explored the distribution. We'll put these two estimators together using a weighted average and define.

$$\hat{\sigma}^2 = \frac{n-1}{n}W + \frac{1}{n}B$$

Notice that if the chains have not yet converged, then $\hat{\sigma}^2$ should overestimate σ^2 because the between chain variance is too large. By itself W will underestimate σ^2 because the chains haven't fully explored the distribution. This suggests

$$\hat{R} = \sqrt{\frac{\hat{\sigma}^2}{W}}$$

is a useful ratio which decreases to 1 as $n \to \infty$. This ratio (Gelman and Rubin, 1992) can be interpreted as the potential reduction in the estimate of $\hat{\sigma}^2$ if we continued sampling.

A closely related is the number of effective samples from the posterior distribution

$$n_{eff} = mn\frac{\hat{\sigma}^2}{B}$$

To demonstrate these, we'll calculate $\hat{R}$ for the above chains

```
# remove the first 200 iterations
trimmed.chains <- chains[201:1000, ]      # Both do the
#trimmed.chains <- chains[-(1:200), ]      # same thing.

# covert matrix to data.frame with 2 columns - Chain, Value
X <- as.data.frame(trimmed.chains)
colnames(X) <- paste('Chain.', 1:ncol(X), sep='')
data <- tidyr::gather(X, key='Chain') %>% # A function from the tidyr package
  mutate(Chain = factor(Chain))
str(data)
```

```
## 'data.frame':    2400 obs. of  2 variables:
##  $ Chain: Factor w/ 3 levels "Chain.1","Chain.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ value: num  3.45 2.77 1.98 1.22 1.72 ...
```

```
# Calculate a few easy things.
n <- nrow(X)
m <- ncol(X)
overall.mean <- mean( data$value )
```

```
# Use the package dplyr to summarise the data frame into W,B
temp <- data %>% group_by(Chain) %>%
   summarise( s2_i = var(value),
              xbar_i = mean(value)) %>%
   summarise( W = mean(s2_i),
              B = n/(m-1) * sum( (xbar_i - overall.mean)^2 ) )
W <- temp$W
B <- temp$B
sigma2.hat <- (n-1)/n * W + (1/n)*B
R <- sqrt(sigma2.hat/W)
n.eff <- m*n * sigma2.hat / B

cbind( W, B, sigma2.hat, R, n.eff )  # print with nice labeling on top!
```

```
##             W        B sigma2.hat        R    n.eff
## [1,] 2.635861 37.45643   2.679386 1.008223 171.6802
```

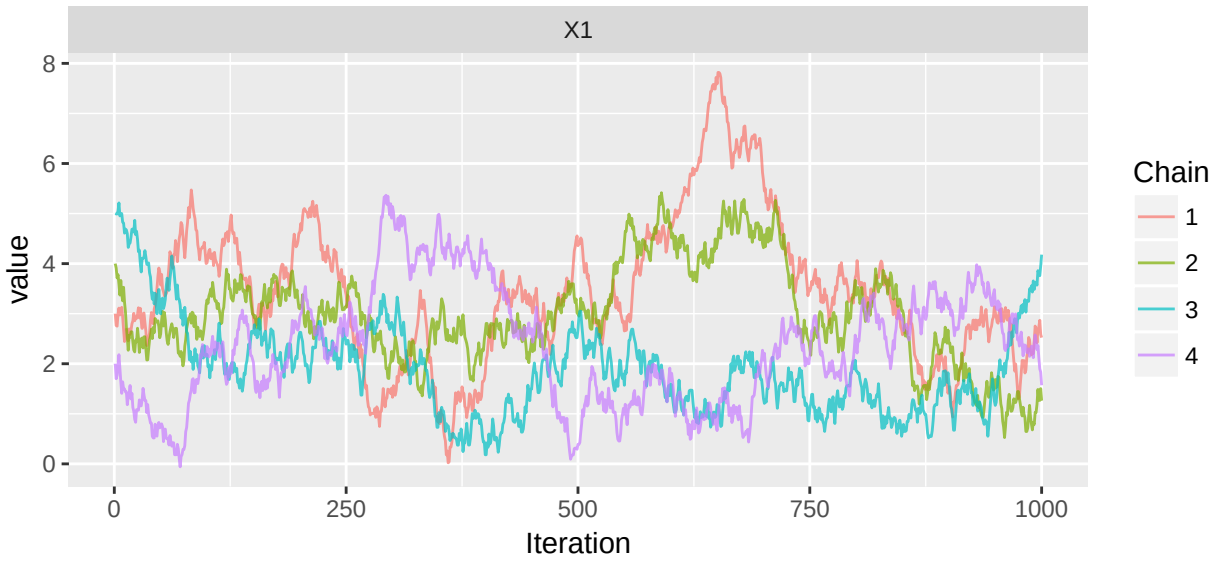
```
# Or just use the Gelman() function in the STA578 package
STA578::Gelman(trimmed.chains)
```

```
## # A tibble: 1 x 6
##   Parameter        W        B sigma2.hat    R.hat    n.eff
##       <fctr>    <dbl>    <dbl>      <dbl>    <dbl>    <dbl>
## 1        x1 2.635861 37.45643   2.679386 1.008223 171.6802
```

A useful rule-of-thumb for assessing convergence is if the $\hat{R}$ value(s) are less than 1.05, then we are close enough. However, this does not imply that we have enough independent draws from the sample to trust our results. In the above example, even though we have $3 \cdot 800 = 2400$ observations, because of the Markov property of our chain, we actually only have about 171 independent draws from the distribution. This suggests that it takes around 14 MCMC steps before we get independence between two observations.

To further explore this idea, lets look at what happens when the chains don't mix as well.

```
chains <- mMCMC(df, list(3,4,5,2),
                rprop=function(x){ return(x+runif(1,-.3,.3)) },
                N=1000, num.chains=4)
trace_plot(chains)
```




```
Gelman(chains)
```

```
## # A tibble: 1 x 6
##   Parameter        W        B sigma2.hat    R.hat    n.eff
##       <fctr>    <dbl>    <dbl>      <dbl>    <dbl>    <dbl>
## 1        X1 1.295232 466.1368   1.760074 1.165713 15.10349
```

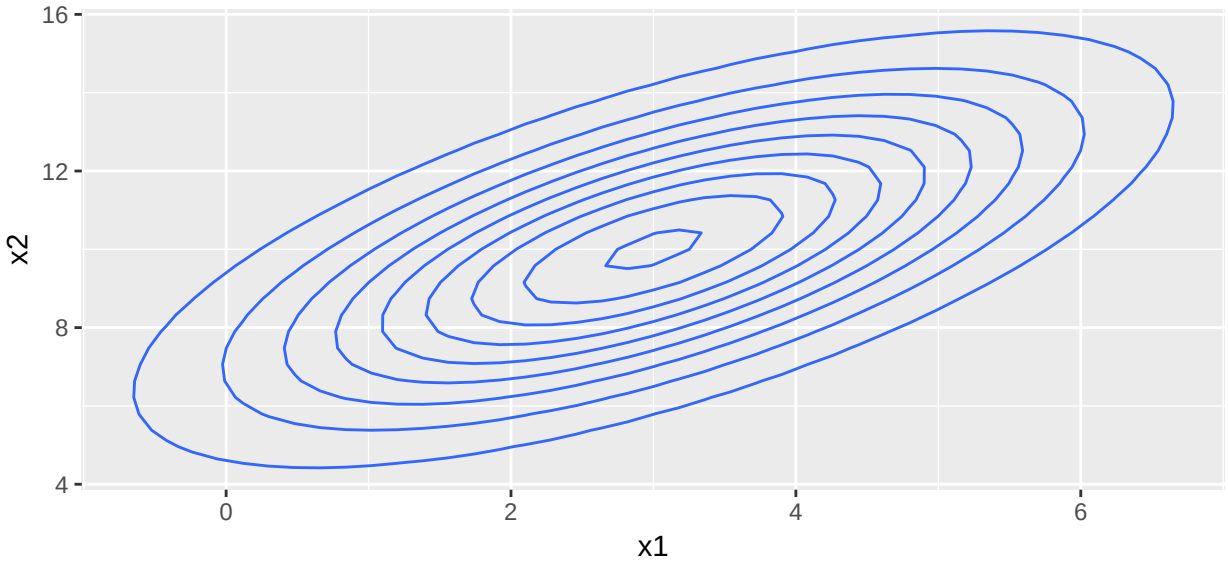
Notice that I didn't have to trim the burn-in part of the chain because I started the chains in the distribution, but because the step-size is too small, the means of the chains are very different from each other and therefore the *between* estimate of the target distribution variance is huge compared to the *within* estimates.

2.5 Multi-variate MCMC

We now consider the case of doing MCMC in multiple dimensions. This is where MCMC is vastly superior to other methods of sampling, however we still have some substantial issues to address. In principle, instead of our proposed value x^*_{i+1} being a slightly perturbed version of x_i in 1-D, we could perturb it in multiple dimensions.

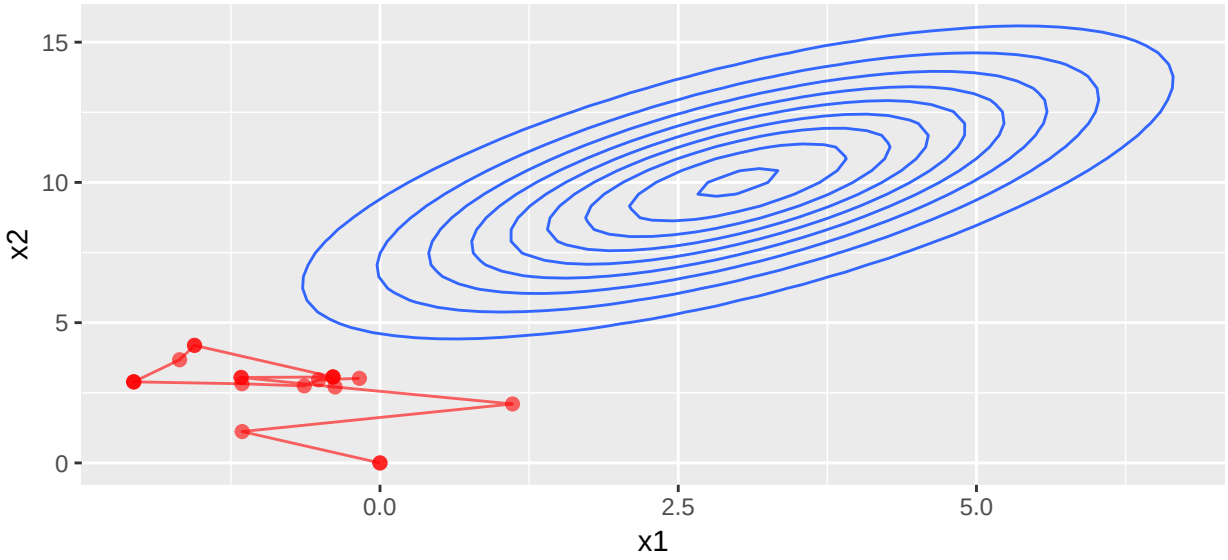
Suppose we wish to sample from the multivariate normal distribution pictured below:

```
dtarget <- function(x){
  dmvnorm(x, mean=c(3,10), sigma=matrix(c(3,3,3,7), nrow=2))
}
x1 <- seq(-6,12,length=101)
x2 <- seq(-11,31, length=101)
contour.data <- expand.grid(x1=x1, x2=x2)
contour.data$Z <- apply(contour.data, MARGIN=1, dtarget)
target.map <- ggplot(contour.data, aes(x=x1, y=x2, z=Z)) +
                stat_contour()
target.map
```

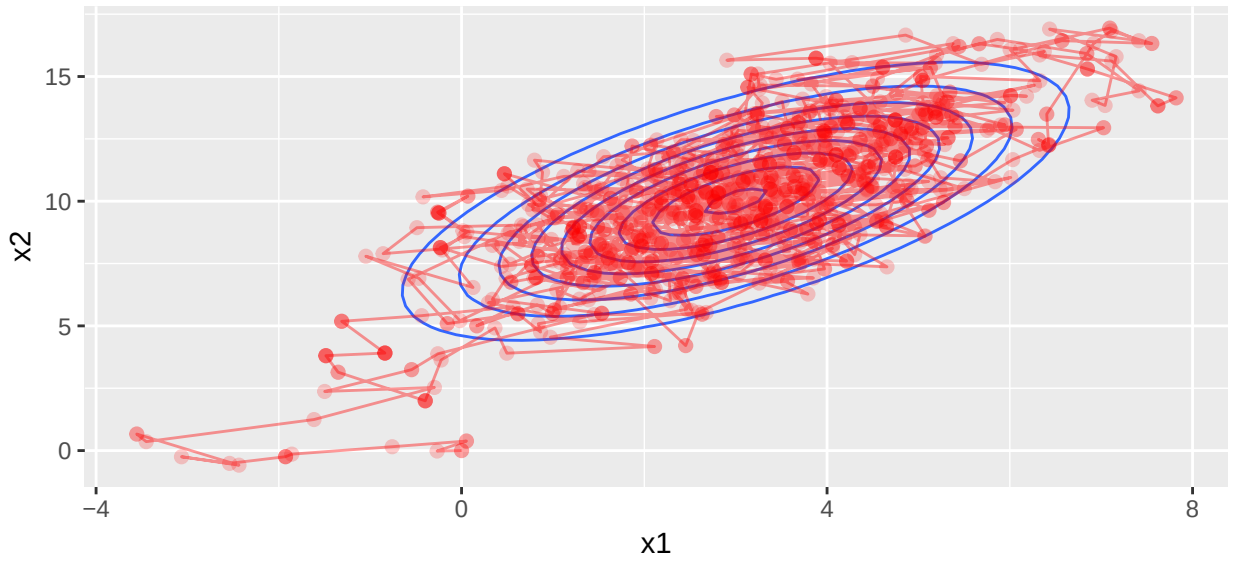


We'll now consider a proposal distribution that is nice and simple, a multivariate normal with no correlation. We start the chain at $(0, 0)$ and it takes awhile to find the distribution.

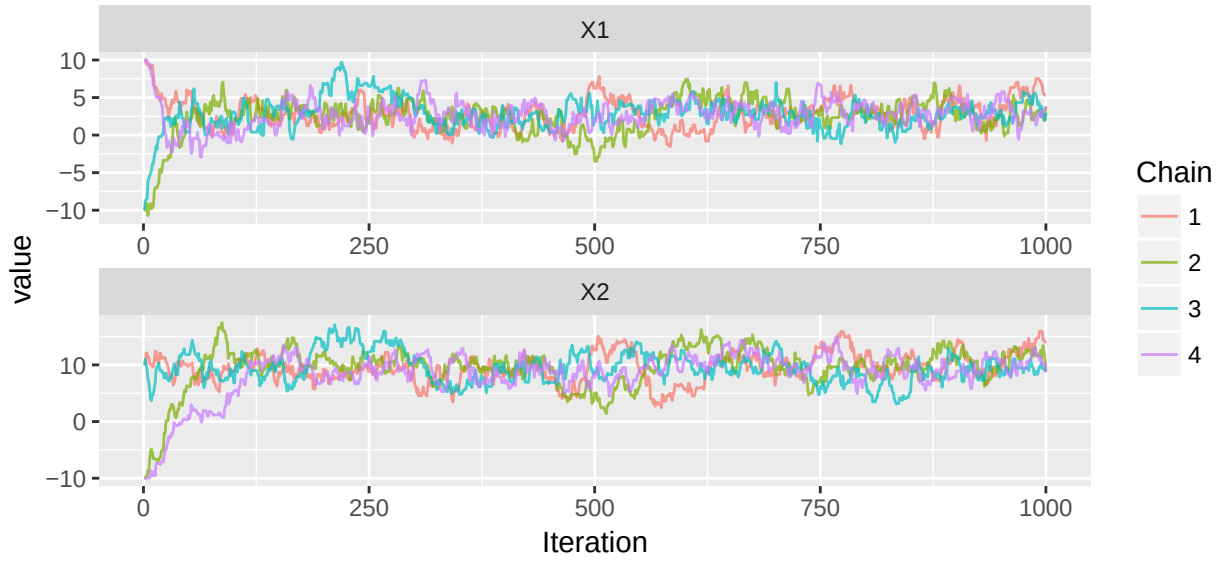
```
rprop <- function(x){
  rmvnorm(1, mean=x, sigma=diag(c(1,1)))
}
start <- c(x1=0, x2=0)
chain <- mMCMC(df=dtarget, start, rprop, N=20, num.chains=1)
target.map +
  geom_path(data=data.frame(chain[[1]]), aes(z=0), color='red', alpha=.6) +
  geom_point(data=data.frame(chain[[1]]), aes(z=0), color='red', alpha=.6, size=2)
```




```
chain <- mMCMC(df=dtarget, start, rprop, N=1000, num.chains=1)
target.map +
  geom_path(data=data.frame(chain[[1]]), aes(z=0), color='red', alpha=.4) +
  geom_point(data=data.frame(chain[[1]]), aes(z=0), color='red', alpha=.2, size=2)
```




```
# four chains, all starting at different places
chains <- mMCMC(df=dtarget,
                start = list( c(10,10), c(-10,-10), c(-10,10), c(10,-10) ),
                rprop, N=1000, num.chains=4)
trace_plot(chains)
```



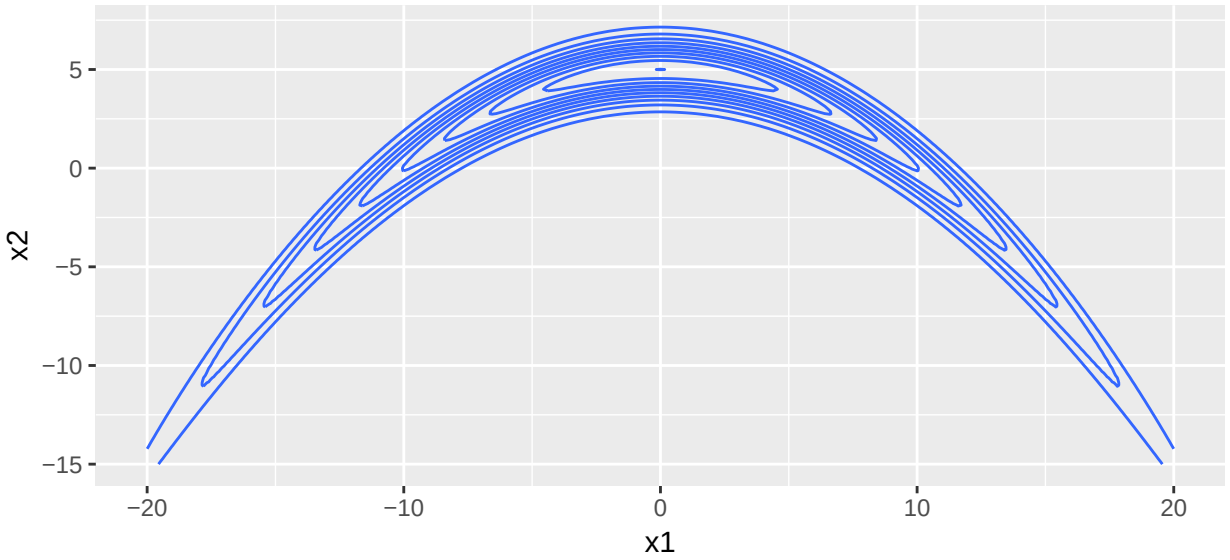

```
Gelman(window(chains, 200, 1000))  # Diagnostics after removing first 200
```

```
## # A tibble: 2 x 6
##   Parameter        W        B sigma2.hat    R.hat    n.eff
##       <fctr>    <dbl>    <dbl>      <dbl>    <dbl>    <dbl>
## 1         X1 3.260507 23.71835   3.286048 1.003909 443.8966
## 2         X2 6.440546 21.65755   6.459544 1.001474 955.6196
```

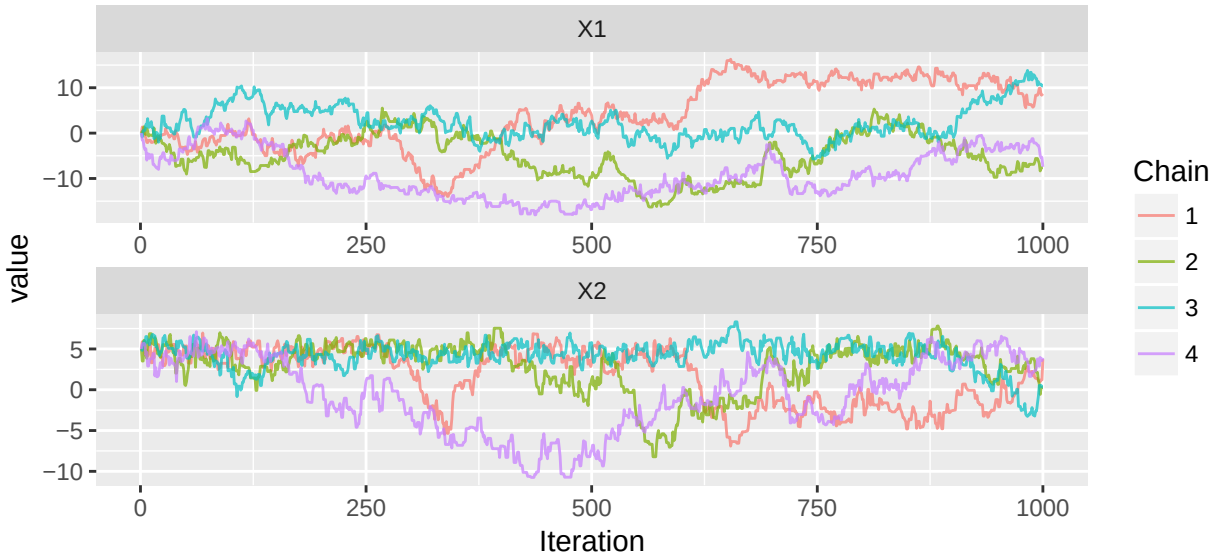
That worked quite nicely, but everything works nicely in the normal case. Lets try a distribution that is a bit uglier, which I'll call the *banana* distribution.

```
dtarget <- function(x){
  B <- .05
  exp( -x[1]^2 / 200 - (1/2)*(x[2]+B*x[1]^2 -100*B)^2 )
```

```
}
x1 <- seq(-20,20, length=201)
x2 <- seq(-15,10, length=201)
contour.data <- expand.grid(x1=x1, x2=x2)
contour.data$Z <- apply(contour.data, MARGIN=1, dtarget)

target.map <- ggplot(contour.data, aes(x=x1, y=x2)) +
  stat_contour(aes(z=Z))
target.map
```

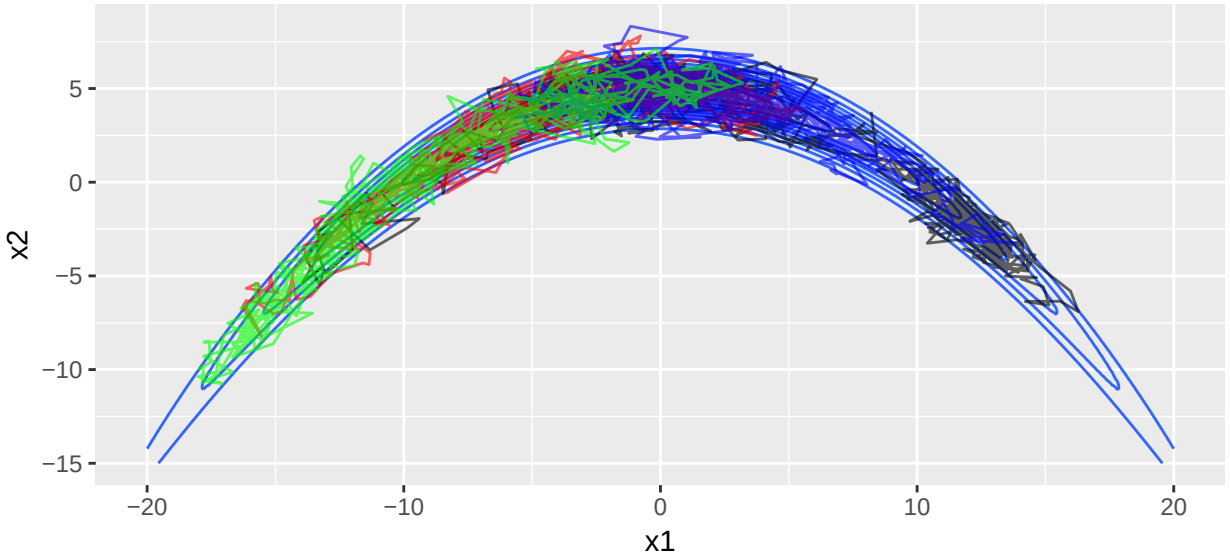



```
# four chains, all starting in the peak of the distribution
chains <- mMCMC(df=dtarget,
                start = c(0,5),
                rprop, N=1000, num.chains=4)
trace_plot(chains)
```



The convergence doesn't look great, and when we overlay the chain paths on the distribution, we see why the convergence isn't good.

```
target.map +
   geom_path(data=data.frame(chains[[1]]), aes(x=X1, y=X2, z=0),
             color='black', alpha=.6) +
  geom_path(data=data.frame(chains[[2]]), aes(x=X1, y=X2, z=0),
             color='red', alpha=.6) +
  geom_path(data=data.frame(chains[[3]]), aes(x=X1, y=X2, z=0),
             color='blue', alpha=.6) +
  geom_path(data=data.frame(chains[[4]]), aes(x=X1, y=X2, z=0),
             color='green', alpha=.6)
```



The chains are having a very hard time moving into the ends of the banana. Unfortunately there is little we can do about this other than run our chains for a very long time. Anything you do to optimize the proposal distribution at the one end of the banana will fail at the other end.

2.6 Hamiltonian MCMC

One of the biggest developments in Bayesian computing over the last decade has been the introduction of Hamiltonian Monte Carlo (usually denoted as HMC). This method works when the distribution we wish to explore is continuous and we (or an algorithm can) evaluate the gradient of the distribution.

Most of the information here comes from Radford Neal's book chapter [http://arxiv.org/pdf/1206.1901.pdf || MCMC using Hamiltonian Dynamics]. We will use his notation and the information in this section is mostly his work, though I've tried not to use his exact verbiage.

Hamiltonian dynamics can be thought of as describing the motion of a hockey puck sliding across a frictionless (possibly non-flat) surface. In this system there are two things to know: the position of the puck (which we'll denote as $\boldsymbol{q}$) and its momentum which is the puck's mass*velocity and which we'll denote as $\boldsymbol{p}$. (I assume there is some horrible physics convention that we are following for this notation). Given this, we can look at the total energy of the puck as it's potential energy $U(\boldsymbol{q})$ which is its height at the current position q and it's kinetic energy $K(p)$. The Hamiltonian part of this is to say that

$$H(\boldsymbol{q},\boldsymbol{p}) = U(\boldsymbol{q}) + K(\boldsymbol{p})$$

remains constant as the puck moves around the surface.

- As the puck slides on a level surface, the potential energy remains the same and so does the kinetic and the puck just slides with constant velocity.
- As the puck slides up an incline, the potential energy increases and the kinetic energy decreases (i.e. the puck slows down).
- As the puck slides down an incline, the potential energy decreases and the kinetic energy increases (i.e. the puck speeds up).

In our case, the position q will correspond the the variables of interest and the momentum p will be auxiliary variables that we add to the problem. Somewhat paradoxically, by making the problem harder (i.e. more variables) we actually make the problem much more tractable. In order to move the puck around the space and stay near the peak of the distribution, we'll define

$$U(\boldsymbol{q}) = -\log f(\boldsymbol{q})$$

so that the edges of the distribution have high potential energy (that is $U(q)$ is a bathtub shaped surface and the puck is making paths around the tub, but never exiting the bathtub). Likewise we'll define the kinetic energy as some standard form involving the velocity and mass.

$$K(\boldsymbol{p}) = \boldsymbol{p}^T\boldsymbol{p}/2m$$

where m is the mass of the puck.

Fortunately Hamiltonian dynamics are well understood and we can approximately solve the system of equations (letting p_i and q_i be the ith component of the $\boldsymbol{p}$ and $\boldsymbol{q}$ vectors)

$$\frac{dq_i}{dt} = m^{-1}p_i$$
$$\frac{dp_i}{dt} = -\frac{\partial U}{\partial q_i}$$

which can be easily discretized to estimate $p_i(t+\epsilon)$ and $q_i(t+\epsilon)$ from a given momentum and location $p_i(t)$ and $q_i(t)$. The most accurate numerical discritization is called "leapfrog" where we take a half-step in p and then a full step in q and a second half step in p.

$$p_i\left(t+\frac{\epsilon}{2}\right) = p_i(t) + \left(\frac{\epsilon}{2}\right)\frac{\partial}{\partial q_i}U(q(t))$$
$$q_i(t+\epsilon) = q_i(t) + \frac{\epsilon}{m}p_i\left(t+\frac{\epsilon}{2}\right)$$
$$p_i(t+\epsilon) = p_i\left(t+\frac{\epsilon}{2}\right) + \left(\frac{\epsilon}{2}\right)\frac{\partial}{\partial q_i}U(q(t+\epsilon))$$

Our plan for MCMC is given a current chain location $\boldsymbol{q}$, we will generate a random momentum vector $\boldsymbol{p}$ and then let the Hamiltonian dynamics run for a little while and then use the resulting location as the proposal $\boldsymbol{q}^*$. Before we do the MCMC, lets first investigate how well the Hamiltonian dynamics work.

To see how well this works, we'll consider our multivariate normal example where `x1` and `x2` are correlated. Given a starting point, we'll generate a random starting point and random momentum and see what happens.

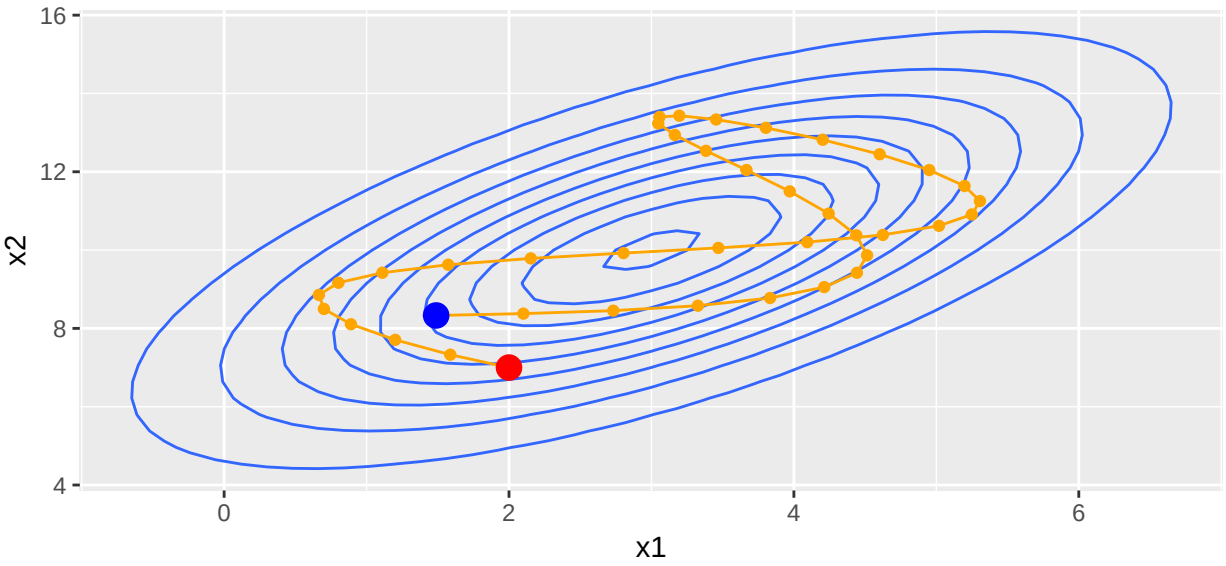
```
dtarget <- function(x, log=FALSE){
  dmvnorm(x, mean=c(3,10), sigma=matrix(c(3,3,3,7), nrow=2), log)
}
x1 <- seq(-6,12,length=101)
x2 <- seq(-11,31, length=101)
contour.data <- expand.grid(x1=x1, x2=x2)
contour.data$Z <- apply(contour.data, MARGIN=1, dtarget)
target.map <- ggplot(contour.data, aes(x=x1, y=x2)) +
                stat_contour(aes(z=Z))
```

```
# set.seed() tarts R's uniform random number generator at a set loction so that
# the rest of this example doesn't change every time I recreate these notes.
set.seed(8675309)
U <- function(x){
  return( -(dtarget(x, log=TRUE)))
}
steps <- HMC.one.step(U, current_q=c(2,7), Eps=.4, L=40, m=1 )
```

```
str(steps)
```

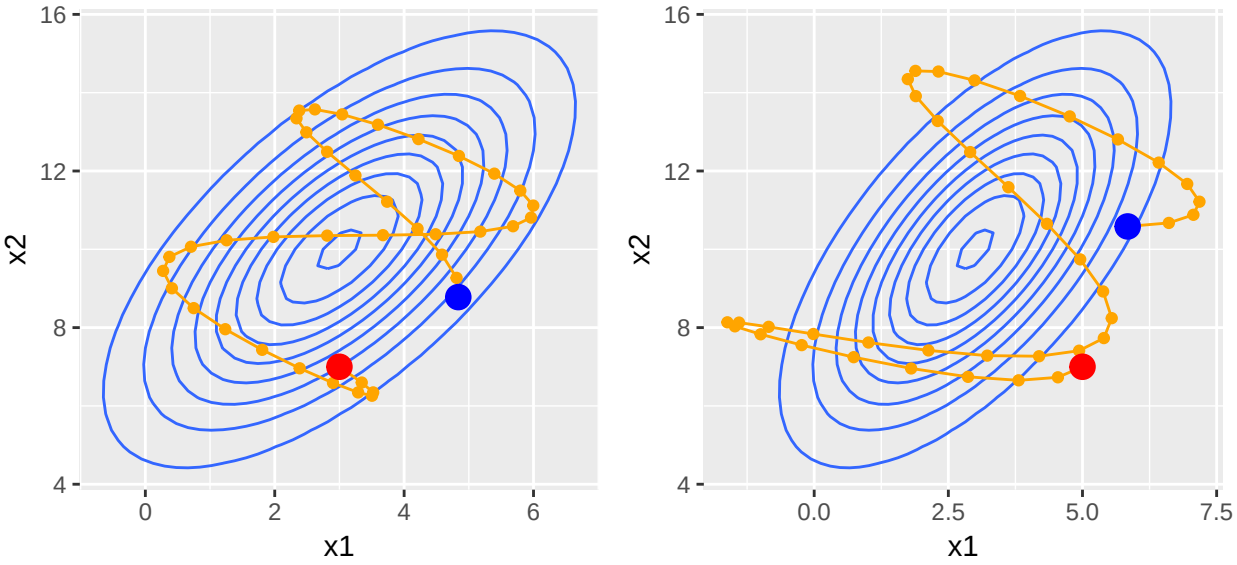
```
## List of 3
##  $ q    : num [1:40, 1:2] 2 1.588 1.201 0.89 0.701 ...
##  $ p    : num [1:40, 1:2] -0.9966 -0.9676 -0.777 -0.4741 -0.0879 ...
##  $ value: num [1, 1:2] 1.49 8.33
```

```
plot_HMC_proposal_path(target.map, steps)
```



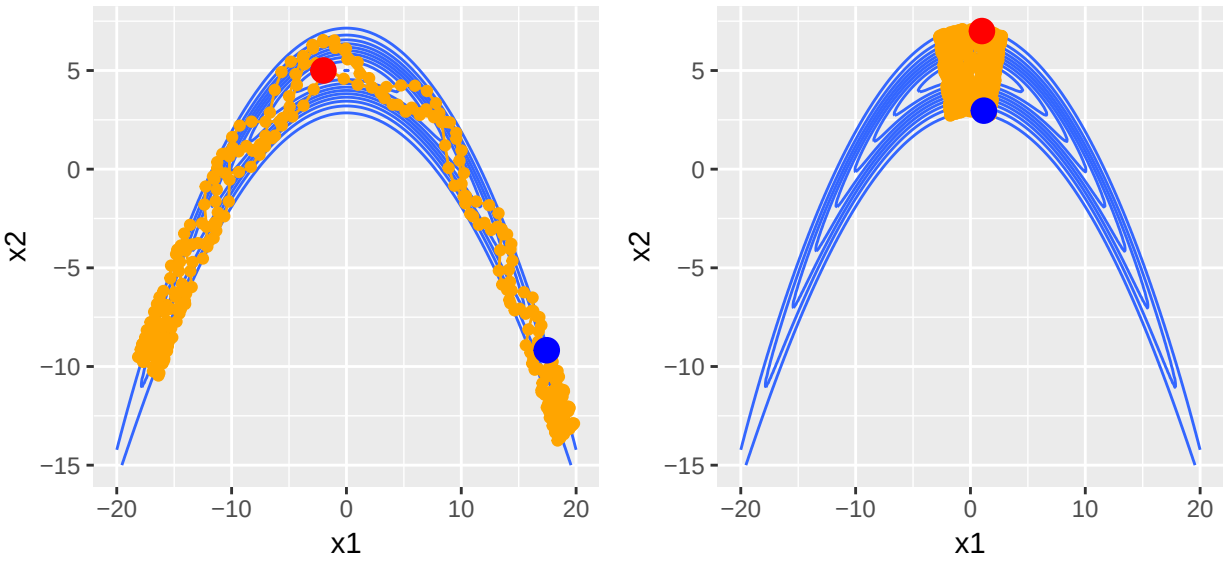
Lets look at a few more of these:

```
set.seed(86709) # so it is reproducible
steps1 <- HMC.one.step(U, current_q=c(3,7), Eps=.4, L=40, m=1 )
steps2 <- HMC.one.step(U, current_q=c(5,7), Eps=.4, L=40, m=1 )
multiplot( plot_HMC_proposal_path(target.map, steps1),
           plot_HMC_proposal_path(target.map, steps2), ncol=2 )
```



Now lets look at some proposed circuits in the banana distribution.

```
dtarget <- function(x, log=FALSE){
  B <- .05
  power <- -x[1]^2 / 200 - (1/2)*(x[2]+B*x[1]^2 -100*B)^2
  if( log ){
    return( power )
  }else{
    exp( power )
  }
}
U <- function(x){
  return( -(dtarget(x, log=TRUE)))
}
x1 <- seq(-20,20, length=201)
x2 <- seq(-15,10, length=201)
contour.data <- expand.grid(x1=x1, x2=x2)
contour.data$Z <- apply(contour.data, MARGIN=1, dtarget)

target.map <- ggplot(contour.data, aes(x=x1, y=x2)) +
  stat_contour(aes(z=Z))

set.seed(427) # so it is reproducible
steps1 <- HMC.one.step(U, current_q=c(-2,5), Eps=.5, L=300, m=1)
steps2 <- HMC.one.step(U, current_q=c(1,7), Eps=.5, L=300, m=1)
multiplot( plot_HMC_proposal_path(target.map, steps1),
           plot_HMC_proposal_path(target.map, steps2), ncol=2)
```



So now our proposal mechanism has three parameters to tune the MCMC with - the step size ϵ, the number of leapfrog steps to take L, and the mass of the particle m.

If we are able to exactly follow the Hamiltonian path, then

$$H(\boldsymbol{q}, \boldsymbol{p}) = H(\boldsymbol{q}^*, \boldsymbol{p}^*)$$

and we should always accept the proposed value. However in practice there are small changes due to numerical issues and we'll add an accept/reject criteria. In the following rule, notice the order of subtraction is opposite of what we normally would do because $U()$ is $-logf()$.

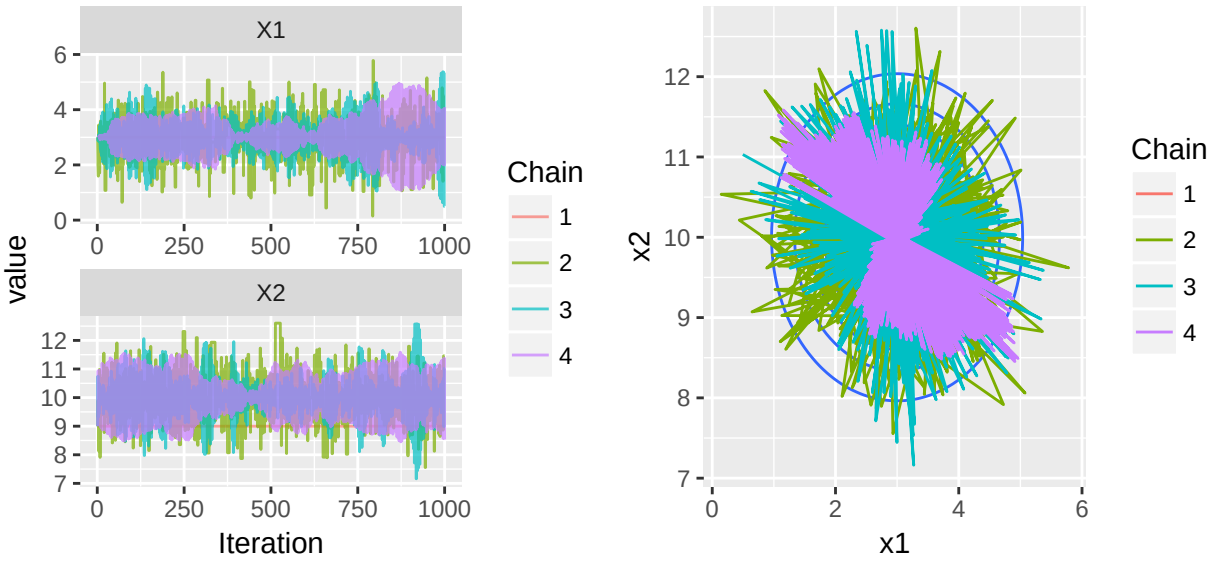
$$\text{accept if } uniform(0,1) < exp\left(H(\boldsymbol{q}, \boldsymbol{p}) - H(\boldsymbol{q}^*, \boldsymbol{p}^*)\right)$$

We will now investigate how sensitive this sampler is to its tuning parameters. We'll first investigate the simple uncorrelated multivariate Normal.

```
dtarget <- function(x){
  dmvnorm(x, mean=c(3,10), sigma=matrix(c(1,0,0,1), nrow=2))
}
x1 <- seq(0,6,length=101)
x2 <- seq(7,13, length=101)
contour.data <- expand.grid(x1=x1, x2=x2)
contour.data$Z <- apply(contour.data, MARGIN=1, dtarget)
target.map <- ggplot(contour.data, aes(x=x1, y=x2)) +
  stat_contour(aes(z=Z))
```

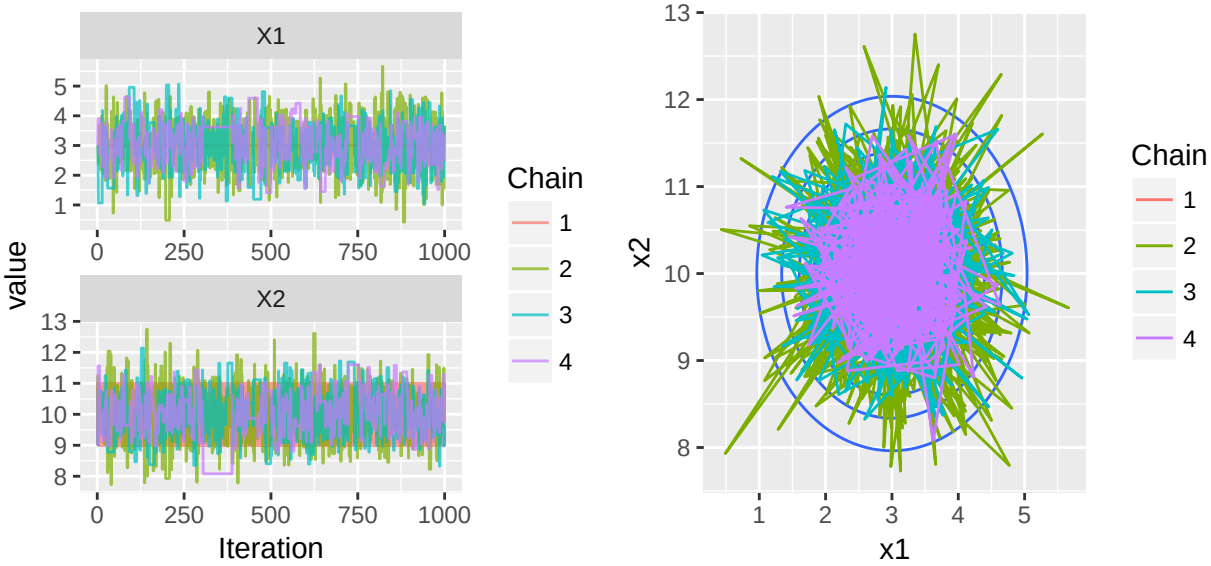
Our first run has chosen the number of leapfrog steps in an unfortunate fashion and our proposed value is often nearly a rotation of 180° from the current value.

```
# Poor behavior of the chain
set.seed(712)
chains <- HMC(dtarget, start=c(3,9), Eps=1, L=7, N=1000, num.chains=4)
multiplot( trace_plot(chains), plot_2D_chains(target.map, chains), ncol=2 )
```



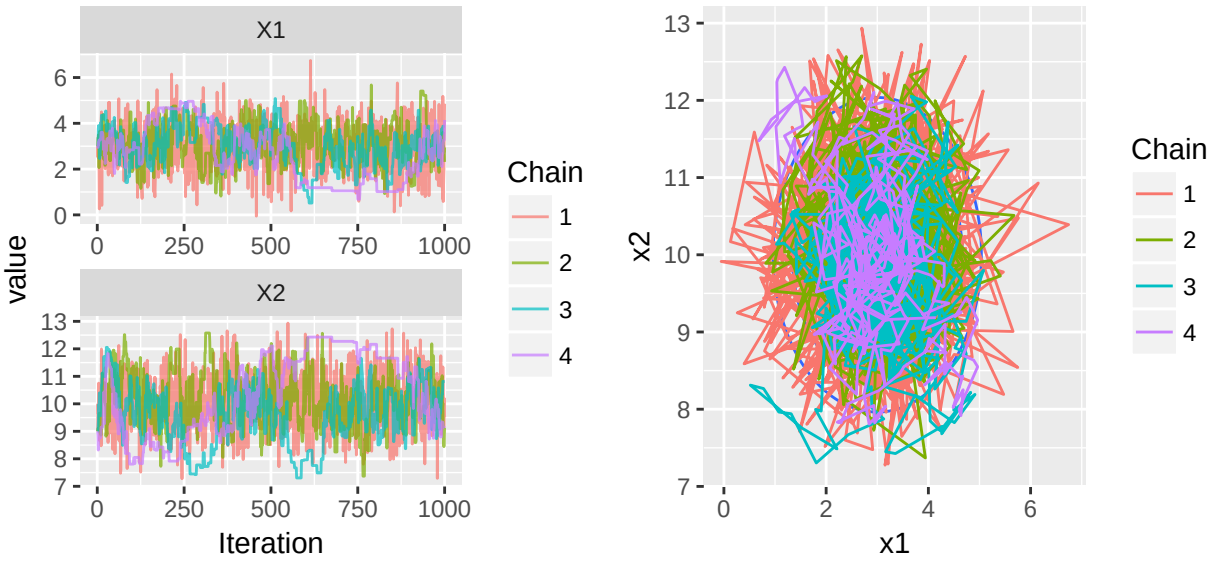
Making an adjustment to the number of leapfrog steps easily fixes this problem, but we have another issue in this next example. Now the within chain variances aren't the same, chain is stuck within a much smaller radius than the other chains.

```
# Poor behavior variances
set.seed(7345)
chains <- HMC(dtarget, start=c(3,9), Eps=1, L=4, N=1000, num.chains=4)
multiplot( trace_plot(chains), plot_2D_chains(target.map, chains), ncol=2 )
```



To try to fix this, lets mess with the mass parameter, step size, and number of leapfrog steps...

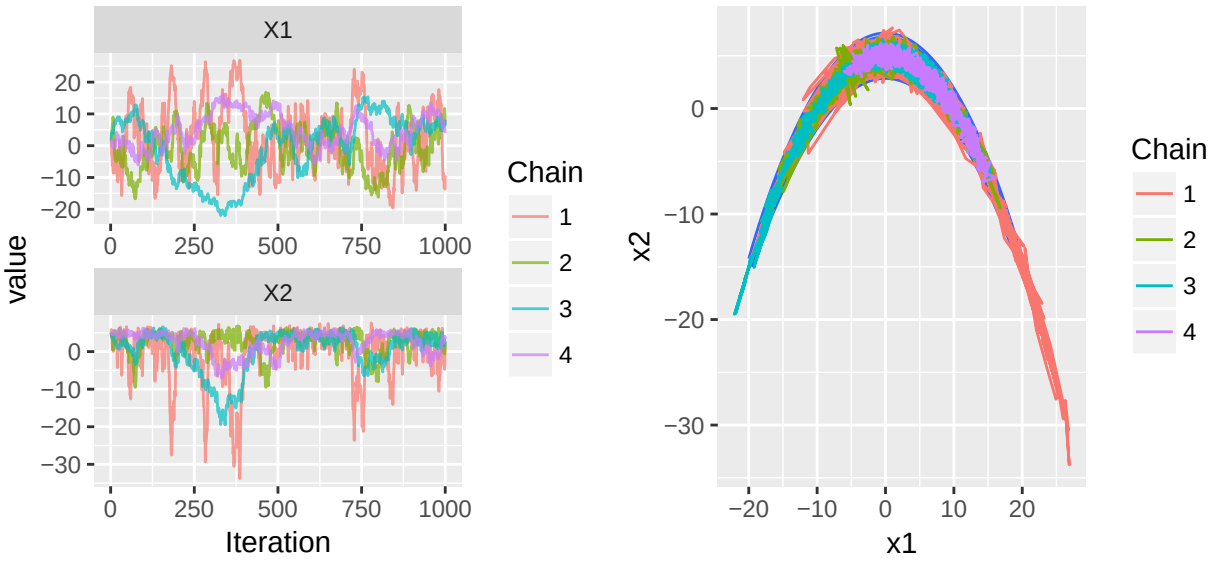
```
set.seed(214)
chains <- HMC(dtarget, start=c(3,9), Eps=.5, L=4, m=.5, N=1000, num.chains=4)
multiplot( trace_plot(chains), plot_2D_chains(target.map, chains), ncol=2 )
```



Next we'll go to the banana distribution

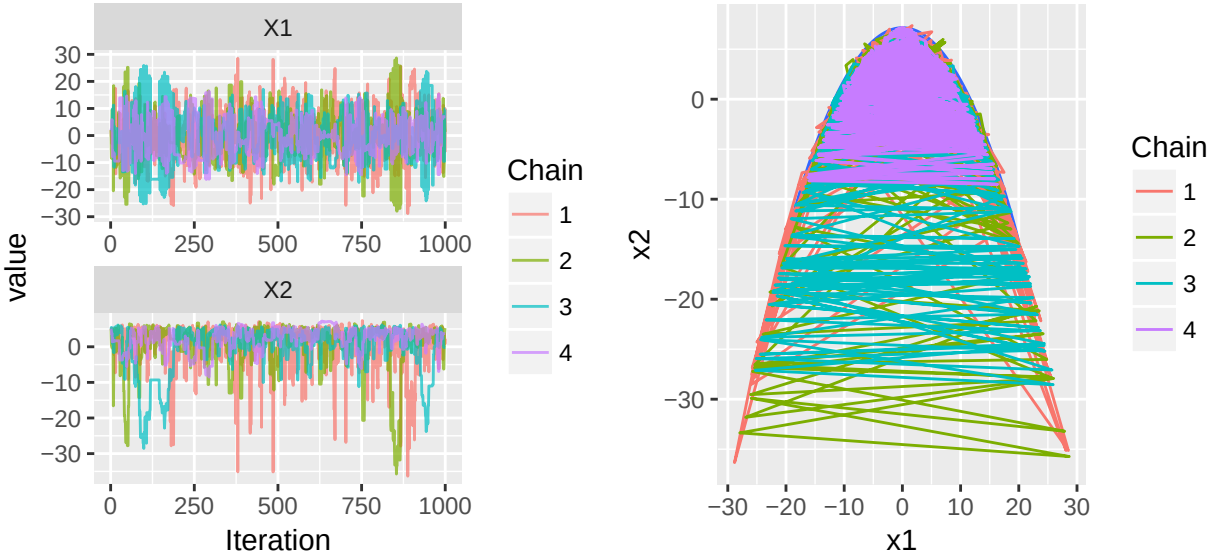
```
dtarget <- function(x){
  B <- .05
  exp( -x[1]^2 / 200 - (1/2)*(x[2]+B*x[1]^2 -100*B)^2 )
}
x1 <- seq(-20,20, length=201)
x2 <- seq(-15,10, length=201)
contour.data <- expand.grid(x1=x1, x2=x2)
contour.data$Z <- apply(contour.data, MARGIN=1, dtarget)
target.map <- ggplot(contour.data, aes(x=x1, y=x2)) +
  stat_contour(aes(z=Z))
```

```
set.seed(21434)
chains <- HMC(dtarget, start=c(2,5), Eps=.5, L=10, m=1)
multiplot( trace_plot(chains), plot_2D_chains(target.map, chains), ncol=2 )
```



That was my first pass with hardly thinking about the tuning parameters. Lets increase the number of leapfrog steps and decrease the mass.

```
set.seed(780)
chains <- HMC(dtarget, start=c(2,5), Eps=.5, L=200, m=.5)
multiplot( trace_plot(chains), plot_2D_chains(target.map, chains), ncol=2 )
```



This is exactly what we want to see. The chains are jumping between the tails and show very little correlation between steps.

These results show that the tuning of the MCMC parameters is very important, but with properly selected values, the MCMC sampler can quite effectively explore distributions. While we haven't considered more interesting examples, the HMC method is quite effective in high dimensional spaces (100s of dimensions) with complicated geometries.

2.7 Exercises

1. The $Beta(\alpha, \beta)$ distribution is a distribution that lives on the interval $[0, 1]$ (for more info, check out the wikipedia page. Write a function called `my.rbeta(n,a,b)` to generate a random sample of size n from the $Beta(a, b)$ distribution using the accept-reject algorithm (*not MCMC*). Your function should mimic the usual `rbeta()` function and return a vector of size n. Generate a random sample of size 1000 from the $Beta(3, 2)$ distribution. Graph the histogram of the sample with the theoretical $Beta(3, 2)$ density superimposed.

   - You may use the function `dbeta()` for the density function.
   - You may generate the maximum height of the distribution by evaluating the density function along a grid of points in $[0, 1]$, then choosing the maximum height and multiplying by something slightly larger than 1 (say 1.10) to get a safe value for M.
   - Because you don't know how many points will be accepted, you might need employ some sort of looping or recursion to make sure you generate n observations. If you accidentally generate more than n data points, you should trim out the excess observations. For a brief introduction to decisions statements and loops in R, see Chapter 11 in A Sufficient Introduction to R.

```
# Some hints about the graphing
library(ggplot2)  # my favorite graphing package

# make a graph of the density
x <- seq(0, 1, length=1001)
density.data <- data.frame( x = x, y=dbeta(x, 3,2) )
```

```
density.plot <- ggplot(density.data, aes(x=x, y=y)) +
  geom_line()
density.plot

# Draw some samples from your function
my.samples <- my.rbeta(1000, 3,2)

# Add a histogram of my.samples to the density.plot
# (this is one of the most obnoxiously hard things to do in ggplot2)
density.plot + geom_histogram(data=data.frame(x=my.samples),
                              aes(x=x, y=..density..),
                              alpha=.5, color='salmon')
```

2. The Rayleigh distribution has density function

$$f(x) = \frac{x}{\sigma^2} e^{-x^2/(2\sigma^2)} \qquad x \geq 0, \;\; \sigma > 0.$$

It is defined for positive values of x and has one parameter (a scale parameter) which we denote as σ. We will consider the problem of using MCMC to draw from this distribution.

a) Create a function `drayleigh(x, scale)` that evaluates $f(x)$ using $\sigma =$ `scale`. Make sure that if you submit a negative value for x, the function returns 0. Likewise, make sure that your function can accept vectors of x values, as in the following example of what you should get.

```
> drayleigh( x=c(-1, 2, 5), scale=3 )
[1] 0.0000000 0.1779416 0.1385290
```

Hint: to handle negative values, you can just round the negative x values values up to zero. The functions ***`pmax()`*** *or* ***`ifelse()`*** *could be very helpful here.*

b) Plot the Rayleigh density function for $\sigma = \{2, 5, 10\}$. *Hint: if you don't want to use a* ***`facet_grid`*** *in ggplot2 you could just make three separate plots and squish them onto the same figure using the* ***`multiplot()`*** *function available in the STA578 package. Check out the help file to see how to use it and some examples.*

c) You will use the `mMCMC()` function that is available in the STA578 library to pull samples from the Rayleigh distribution. Consider the case where scale=2 and we will use a proposal distribution that is Uniform(-0.1, +0.1) about the current chain value. Run four chains for 1000 steps each. Assess convergence and comment on your results. Start each chain at $x = 1$. *Hint: Because you can't throw the scale parameter into* ***`mMCMC()`****, you'll need to make a target distribution function that looks like* ***`dtarget = function(x){drayleigh(x,scale=2)}`*** *which just hard codes the scale you are currently interested in.*

d) Repeat part (c) but now consider a proposal distribution that is Uniform(-1, 1) about the current chain value. Assess convergence and comment on your results.

e) Repeat part (c) but now consider a proposal distribution that is Uniform(-10, 10) about the current chain value. Assess convergence and comment on your results.

f) Repeat part (c) but now consider a proposal distribution that is Uniform(-50, 50) about the current chain value. Assess convergence and comment on your results.

g) Repeat steps (c)-(f) but now the target distribution to sample from is the Rayleigh distribution with scale=15.

h) Comment on the need to tune the width of the proposal distribution relative to the distribution of interest.

Chapter 3

Overview of Statistical Learning

Chapter 2 of *Introduction to Statistical Learning* is very conceptual and there isn't much code to mess with, at least until the end. At the end of the chapter, they introduce the machine learning algorithm K-nearest Neighbors but don't mention anything about how to do an analysis with it.

```
library(ggplot2)
library(dplyr)
library(caret)  # for the train() function
```

There are many packages in R that implement a wide variety of machine learning techniques, and each package works in slightly different ways based on the peculiarities of the individual package authors. Unfortunately makes working with these algorithms a pain because we have to learn how each package does things.

The R package `caret` attempts to resolve this by providing a uniform interface to the most widely used machine learning packages. Whenever possible we will use the `caret` interface, but there will be times where we have to resort to figuring out the peculiarities of specific packages.

If you are interested in more information about how the `caret` package works, and what methods are available, I've found the vignette and manual very helpful. If video tutorials are your thing, there is a DataCamp course and the introductory chapter is free.

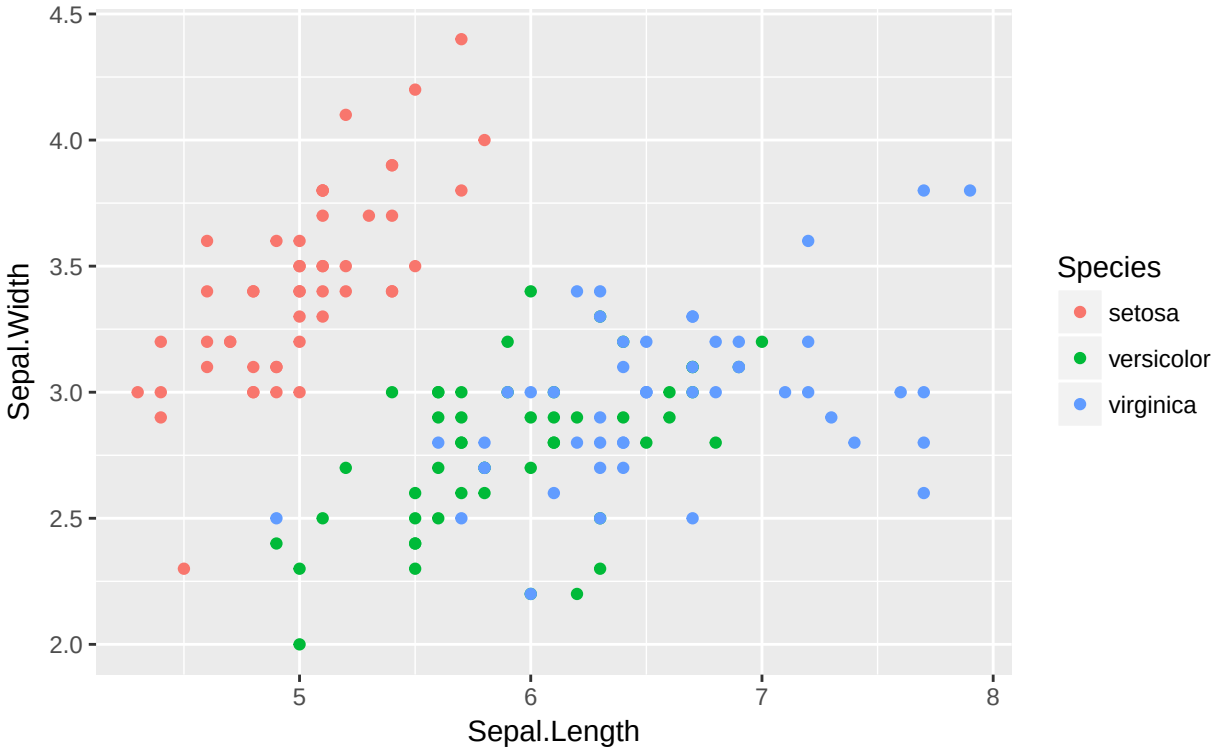
3.1 K-Nearest Neighbors

K-nearest neighbors (KNN) is a technique in machine learning that is extremely simple to implement and extremely easy to understand. We will use this as a prototypical machine learning technique for supervised learning. The idea is that if we wish to predict a response, $\hat{y}_0$ given some covariate(s) x_0, then we look at the k nearest data points to x_0 and take the average of those nearest y values (or in the classification setting, the group with highest representation within the k nearest neighbors).

3.1.1 KNN for Classification

Suppose we consider the `iris` dataset and we wish to provide a classifier for iris species based on the sepal length and sepal width.

```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species)) +
  geom_point()
```



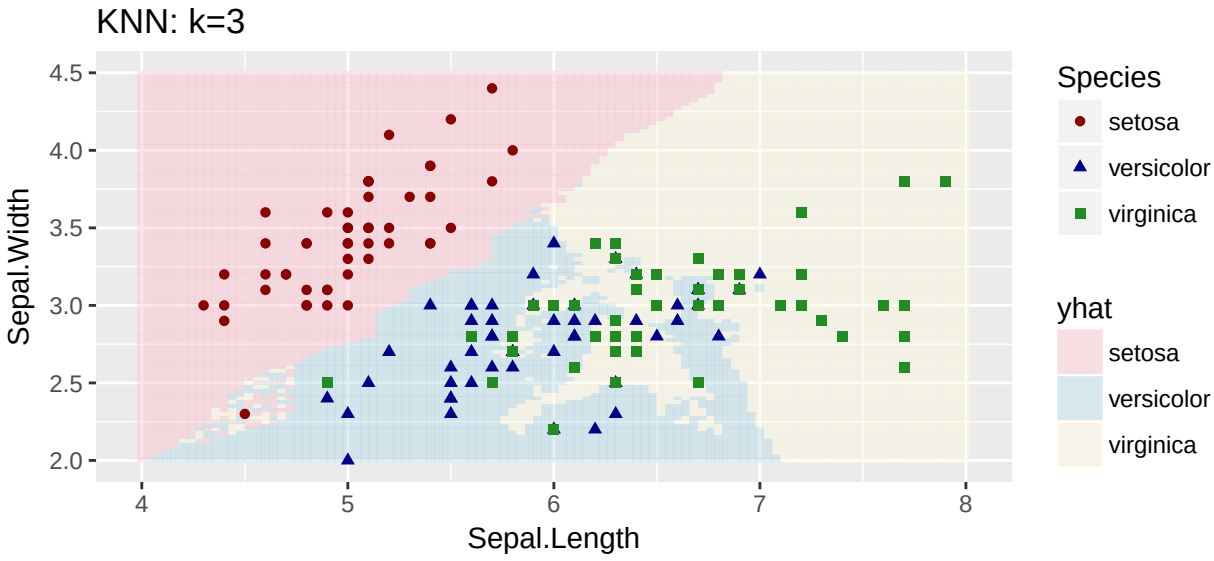
Clearly anything in the upper left should be Setosa, anything in the lower left should be Versicolor, and anything on the right half should be Virginica.

So lets consider KNN.

```
model <- train(Species ~ Sepal.Length + Sepal.Width, data=iris,
               method='knn', tuneGrid=data.frame(k=3))

# expand.grid() makes a data.frame() with all possible combinations
Pred.grid <- expand.grid( Sepal.Length = seq(4, 8, length=101),
                          Sepal.Width  = seq(2, 4.5, length=101) )

Pred.grid$yhat <- predict(model, newdata=Pred.grid)
```

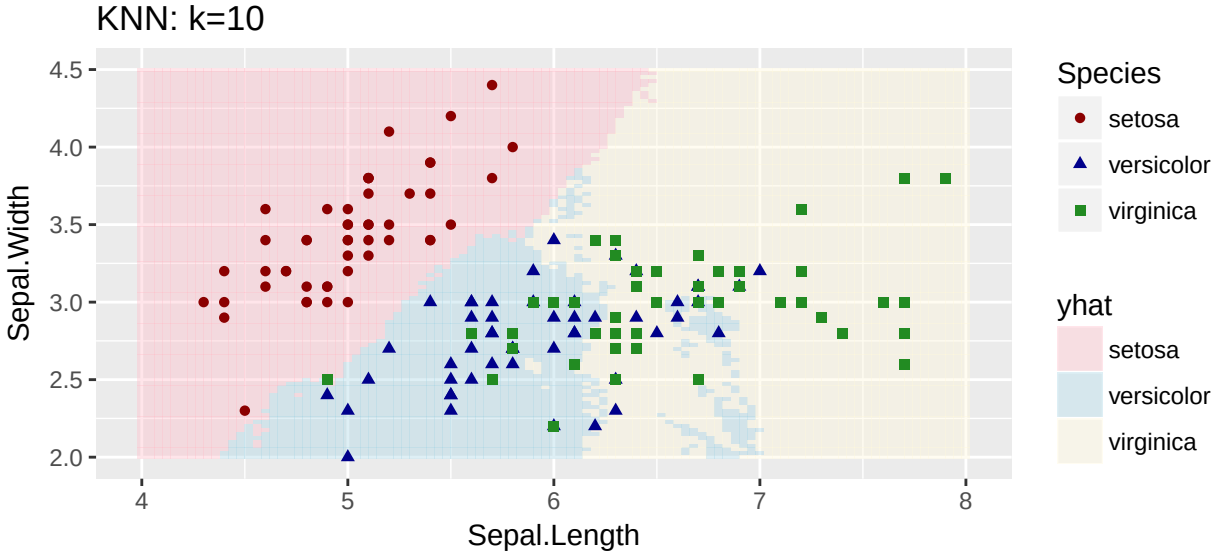
Now that we have the predicted classifications for the grid, lets graph it.

```
ggplot(Pred.grid, aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_tile(aes(fill=yhat), alpha=.4) +
  geom_point(data=iris, aes(color=Species, shape=Species)) +
  scale_color_manual(values=c('dark red', 'dark blue','forestgreen')) +
  scale_fill_manual(values=c('pink', 'light blue','cornsilk')) +
  ggtitle('KNN: k=3')
```



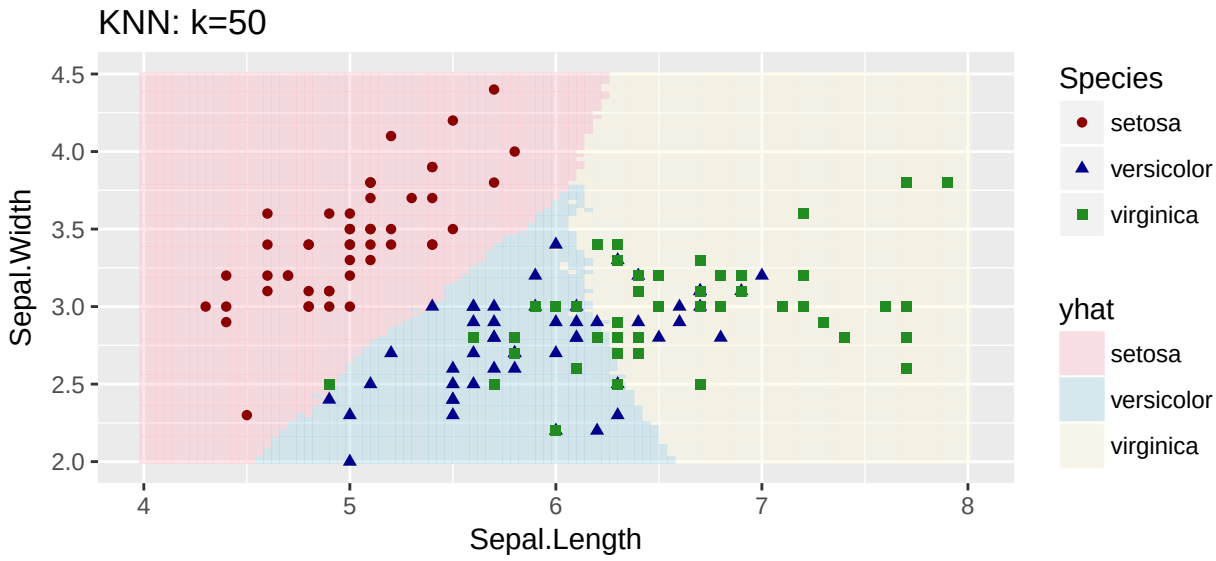
Now we want to see what happens as we vary k.

```
Current.K <- 10
model <- train(Species ~ Sepal.Length + Sepal.Width, data=iris,
               method='knn', tuneGrid=data.frame(k=Current.K))
Pred.grid$yhat <- predict(model, Pred.grid)

ggplot(Pred.grid, aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_tile(aes(fill=yhat), alpha=.4) +
  geom_point(data=iris, aes(color=Species, shape=Species)) +
  scale_color_manual(values=c('dark red', 'dark blue','forestgreen')) +
  scale_fill_manual(values=c('pink', 'light blue','cornsilk')) +
  ggtitle(paste('KNN: k=', Current.K, sep=''))
```



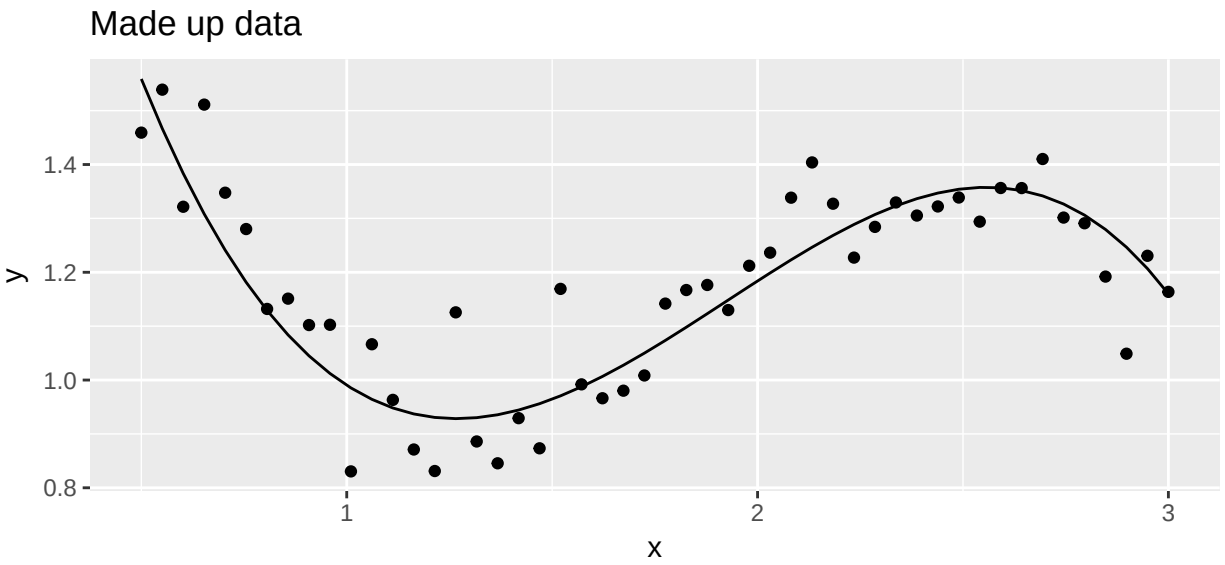

```
Current.K <- 50
model <- train(Species ~ Sepal.Length + Sepal.Width, data=iris,
               method='knn', tuneGrid=data.frame(k=Current.K))
Pred.grid$yhat <- predict(model, Pred.grid)
```

```
ggplot(Pred.grid, aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_tile(aes(fill=yhat), alpha=.4) +
  geom_point(data=iris, aes(color=Species, shape=Species)) +
  scale_color_manual(values=c('dark red', 'dark blue','forestgreen')) +
  scale_fill_manual(values=c('pink', 'light blue','cornsilk')) +
  ggtitle(paste('KNN: k=', Current.K, sep=''))
```



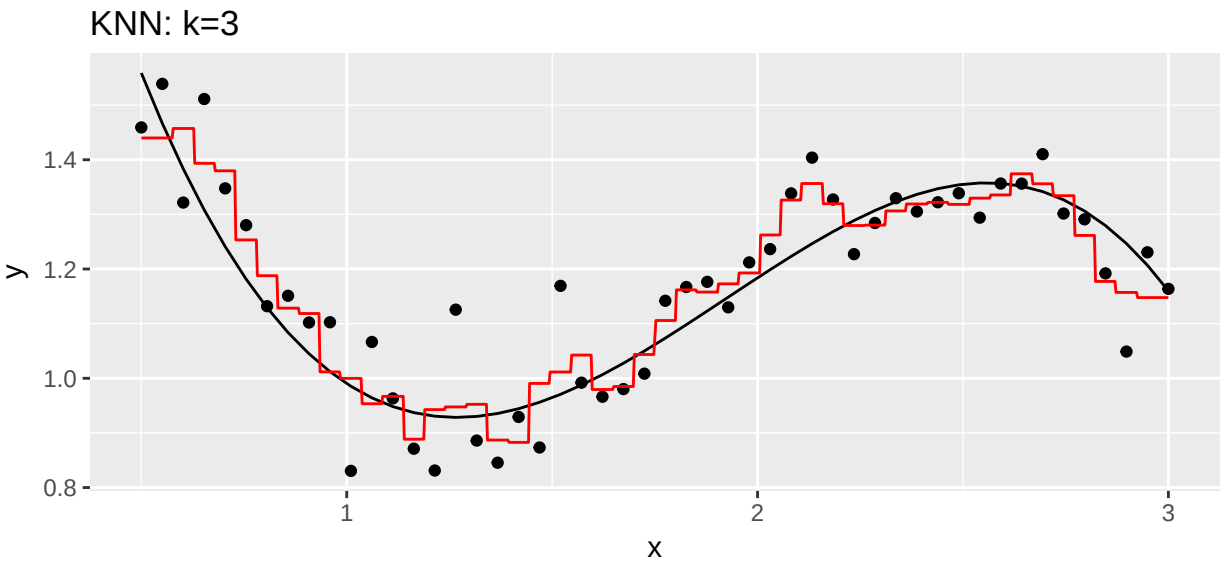
3.1.2 KNN for Regression

```
set.seed(8675309)
n <- 50
data <- data.frame( x=seq(.5,3, length=n) ) %>%
  mutate( Ey = 2 - 4*x + 3*x^2 - .55*x^3 + cos(x),
          y = Ey + rnorm(n, sd=.1))
ggplot(data, aes(x=x)) +
  geom_line(aes(y=Ey)) +
  geom_point(aes(y=y)) +
  labs(main='foo', y='y', title='Made up data')
```



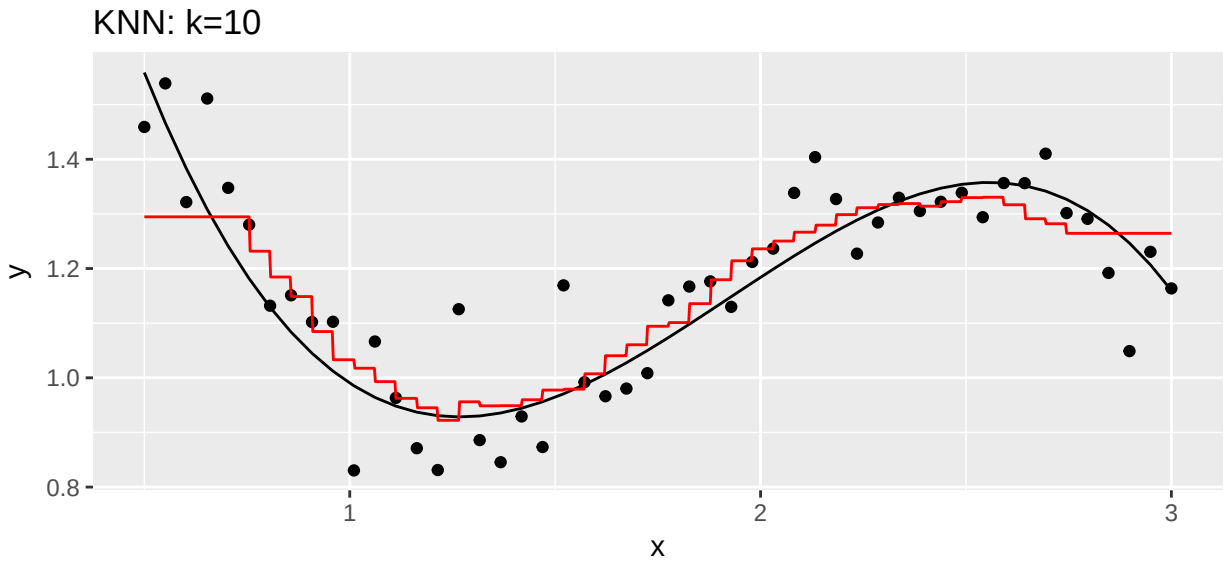

```
# values of x where we wish to make a prediction
Pred.grid <- data.frame(x=seq(0.5, 3, length=1001))
```

```
Current.K <- 3
model <- train(y~x, data=data, method='knn', tuneGrid=data.frame(k=Current.K))
Pred.grid$yhat <- predict(model, Pred.grid)

ggplot(data, aes(x=x)) +
  geom_line(aes(y=Ey)) +
  geom_point(aes(y=y)) +
  geom_line( data=Pred.grid, aes(y=yhat), color='red') +
  labs(main='foo', y='y', title=paste('KNN: k=', Current.K, sep=''))
```

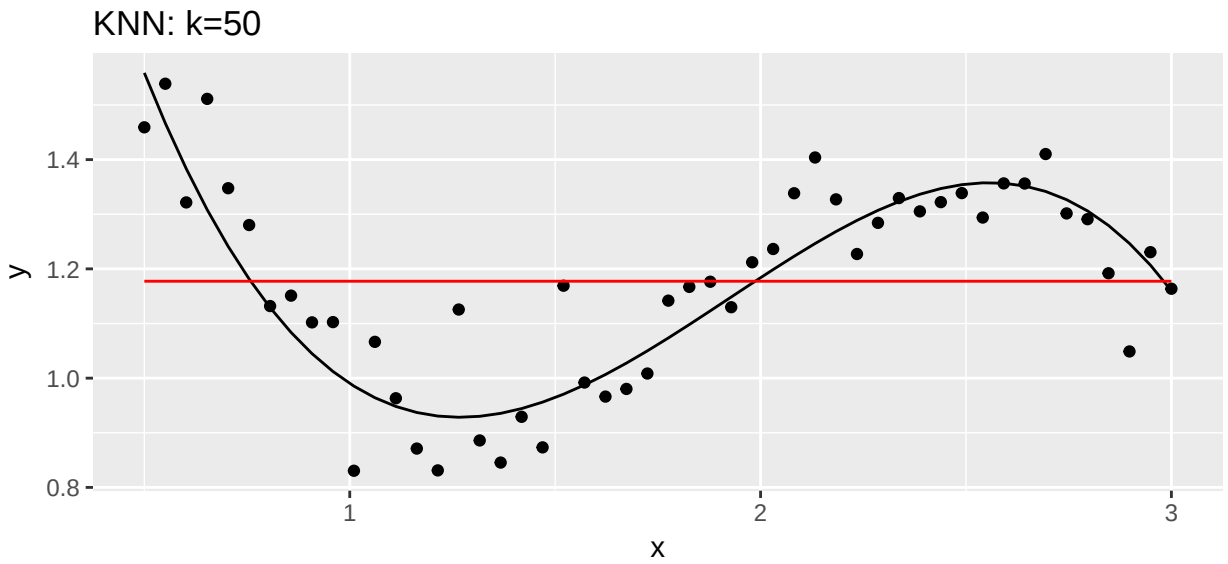



```
Current.K <- 10
model <- train(y~x, data=data, method='knn', tuneGrid=data.frame(k=Current.K))
Pred.grid$yhat <- predict(model, Pred.grid)

ggplot(data, aes(x=x)) +
  geom_line(aes(y=Ey)) +
```

```
  geom_point(aes(y=y)) +
  geom_line( data=Pred.grid, aes(y=yhat), color='red') +
  labs(main='foo', y='y', title=paste('KNN: k=', Current.K, sep=''))
```




```
Current.K <- 50
model <- train(y~x, data=data, method='knn', tuneGrid=data.frame(k=Current.K))
Pred.grid$yhat <- predict(model, Pred.grid)

ggplot(data, aes(x=x)) +
  geom_line(aes(y=Ey)) +
  geom_point(aes(y=y)) +
  geom_line( data=Pred.grid, aes(y=yhat), color='red') +
  labs(main='foo', y='y', title=paste('KNN: k=', Current.K, sep=''))
```



3.2 Splitting into a test and training sets

We continue to consider the Regression problem but we now consider splitting the data into Training and Test sets and comparing the Root Mean Squared Error (RMSE) of Test sets for various values of K.

```
Train <- data %>% sample_frac(.5)
Test  <- setdiff(data, Train)

K <- 5
model <- train(y ~ x, data=Train, method='knn', tuneGrid=data.frame(k=K) )
Test$yhat <- predict(model, Test)
Test %>% mutate(resid = y-yhat) %>%
  summarise( Test.RMSE = sqrt(mean( resid^2 )), K = K)
```

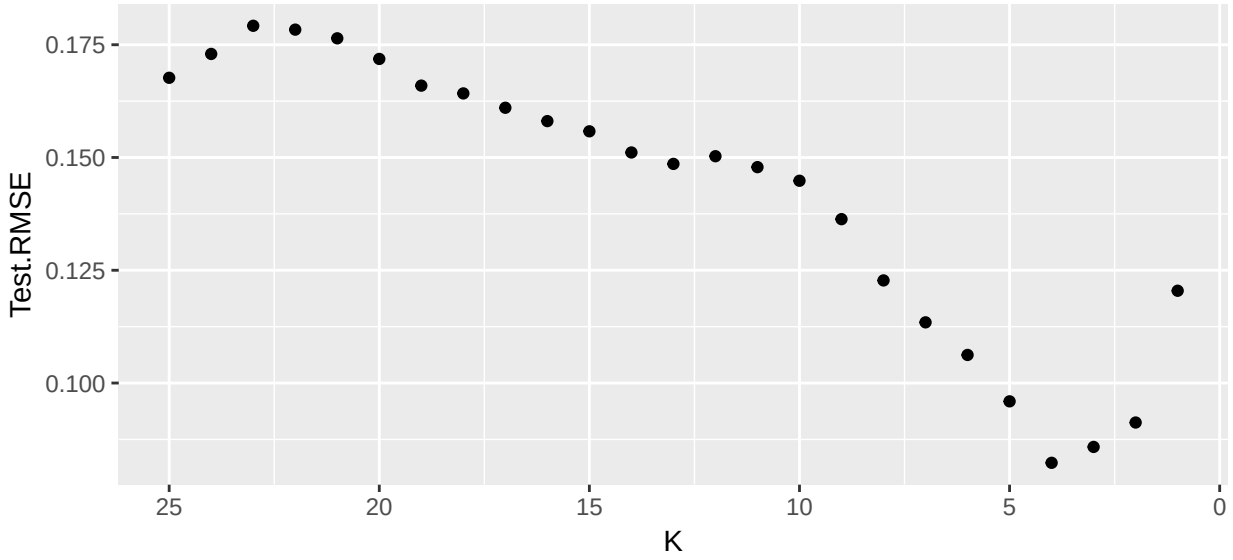
```
##    Test.RMSE K
## 1 0.09594326 5
```

Now that we see how to calculate the Test Mean Squared Error, lets look at a range of values for K.

```
Results <- NULL
for(K in 1:25){
  model <- train(y ~ x, data=Train, method='knn', tuneGrid=data.frame(k=K) )
  Test$yhat <- predict(model, Test)

  Temp <- Test %>% mutate(resid = y-yhat) %>%
      summarise( Test.RMSE = sqrt(mean( resid^2 )), K = K)
  Results <- rbind(Results, Temp)
}
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.
```

```
ggplot(Results, aes(x=K, y=Test.RMSE)) +
  geom_point() +
  scale_x_reverse() # so model complexity increases along x-axis.
```



3.3 Exercises

1. ISL 2.1: For each part below, indicate whether we would generally expect the performance of a flexible statistical learning method to be better or worse than an inflexible method. Justify your answer.
   a) The sample size n is extremely large, and the number of predictors p is small.
   b) The number of predictors p is extremely large, and the number of observations n is small.
   c) Relationship between the predictors and response is highly non-linear.
   d) The variance of the error terms, i.e. $\sigma^2 = Var(\epsilon)$, is extremely high.
2. ISL 2.2: Explain whether each scenario is a classification or regression problem, and indicate whether we are most interested in inference or prediction. Finally, provide n and p.
   a) We collect a set of data on the top 500 firms in the US. For each firm we record profit, number of employees, industry and the CEO salary. We are interested in understanding which factors affect CEO salary.
   b) We are considering launching a new product and wish to know whether it will be a `success` or `failure`. We collect data on 20 similar products that were previously launched. For each product we have record whether it was a success or failure, price charged for the product, marketing budget, competition price, and ten other variables.
   c) We are interested in predicting the % change in the US dollar in relation to the weekly changes in the world stock markets. Hence we collect weekly data for all of 2012. For each week we record the % change in the dollar, the % change in the US market, the % change in the British market, and the % change in the German market.
3. ISL 2.3: We new revisit the bias-variance decomposition.
   a) Provide a sketch of typical (squared) bias, variance, training error, test error, and Bayes (or irreducible) error curves, on a single plot, as we go from less flexible statistical learning methods towards more flexible approaches. The x-axis should represent the amount of flexibility in the method, and the y-axis should represent the values for each curve. There should be five curves. Make sure to label each one.
   b) Explain why each of the five curves has the shape displayed in part (a).

Chapter 4

Classification with LDA, QDA, and KNN

```
library(caret)
library(MASS)     # lda function used by caret
library(dplyr)    # data frame manipulations
library(ggplot2)  # plotting
library(STA578)   # for my multiplot() function
library(pROC)     # All the ROC stuff
```

In this chapter we look at several forms of classification. In particular we will focus on the classical techniques of logistic regression, Linear Discriminent Analysis and Quadratic Discriminent Analysis.

4.1 Logistic Regression

We won't cover the theory of logistic regression here, but you can find it elsewhere.

The dataset `faraway::wbca` comes from a study of breast cancer in Wisconsin. There are 681 cases of potentially cancerous tumors of which 238 are actually malignant (ie cancerous). Determining whether a tumor is really malignant is traditionally determined by an invasive surgical procedure. The purpose of this study was to determine whether a new procedure called 'fine needle aspiration', which draws only a small sample of tissue, could be effective in determining tumor status.

```
data('wbca', package='faraway')

# clean up the data
wbca <- wbca %>%
  mutate(Class = ifelse(Class==0, 'malignant', 'benign')) %>%
  dplyr::select(Class, BNucl, UShap, USize)

# Fit the model
# Malignant is considered a success
model <- glm( I(Class=='malignant') ~ ., data=wbca, family='binomial' )

# Get the response values
# type='response' gives phat values which live in [0,1]
# type='link' gives the Xbeta values whice live in (-infinity, infinity)
```

```
wbca <- wbca %>%
  mutate(phat = predict(model, type='response'),
         yhat = ifelse(phat > .5, 'malignant', 'benign'))

# Calculate the confusion matrix
table( Truth=wbca$Class, Predicted=wbca$yhat )
```

```
##            Predicted
## Truth       benign malignant
##   benign       432        11
##   malignant     15       223
```

As usual we can calculate the summary tables...

```
summary(model)
```

```
##
## Call:
## glm(formula = I(Class == "malignant") ~ ., family = "binomial",
##     data = wbca)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.8890  -0.1409  -0.1409   0.0287   2.2284
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -6.35433    0.54076 -11.751  < 2e-16 ***
## BNucl        0.55297    0.08041   6.877 6.13e-12 ***
## UShap        0.62583    0.17506   3.575 0.000350 ***
## USize        0.56793    0.15910   3.570 0.000358 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 881.39  on 680  degrees of freedom
## Residual deviance: 148.43  on 677  degrees of freedom
## AIC: 156.43
##
## Number of Fisher Scoring iterations: 7
```

From this table, we see that for a breast tumor, the larger values of `BNucl`, `UShap`, and `USize` imply a greater probability of it being malignant. So for a tumor with

```
##   BNucl UShap USize
## 1     2     1     2
```

We would calculate

$$\begin{aligned} X\hat{\beta} &= \hat{\beta}_0 + 2 \cdot \hat{\beta}_1 + 1 \cdot \hat{\beta}_2 + 2 \cdot \hat{\beta}_3 \\ &= -6.35 + 2 * (0.553) + 1 * (0.626) + 2 * (0.568) \\ &= -3.482 \end{aligned}$$

and therefore

$$\hat{p} = \frac{1}{1 + e^{-X\hat{\beta}}} = \frac{1}{1 + e^{3.482}} = 0.0297$$

```
newdata = data.frame( BNucl=2, UShap=1, USize=2 )
predict(model, newdata=newdata)
```

```
##         1
## -3.486719
```

```
predict(model, newdata=newdata, type='response')
```

```
##         1
## 0.0296925
```

So for a tumor with these covariates, we would classify it as most likely to be benign.

4.2 ROC Curves

In the medical scenario where we have to decide if a tumor is malignant of benign, we shouldn't treat the misclassification errors as being the same. If we incorrectly identify a tumor as malignant when it is not, that will cause a patient to undergo a somewhat invasive surgury to remove the tumor. However if we incorrectly identify a tumor as being benign, then the cancerous tumor will likely grow and eventually kill the patient. While the first error is regretable, the second is far worse.

Given that reasoning, perhaps we shouldn't use the rule: If $\hat{p} >= 0.5$ classify as malignant. Instead perhaps we should use $\hat{p} >= 0.3$ or $\hat{p} >= 0.05$.

Whatever decision rule we make, we should consider how many of each type of error we make. Consider the following confusion maatrix:

	Predict Positive	Predict Negative	Total
Positive	True Pos (TP)	False Neg (FN)	P
Negative	False Pos (FP)	True Neg (TN)	N
Total	P^*	N^*	

where P is the number of positive cases, N is the number of negative cases, P^* is the number of observations predicted to be positive, and N^* is the number predicted to be negative.

Quantity	Formula	Synonyms
False Positive Rate	FP/N	Type I Error; 1-Specificity
True Positive Rate	TP/P	Power; Sensitivity; Recall
Pos. Pred. Value	TP/P^*	Precision

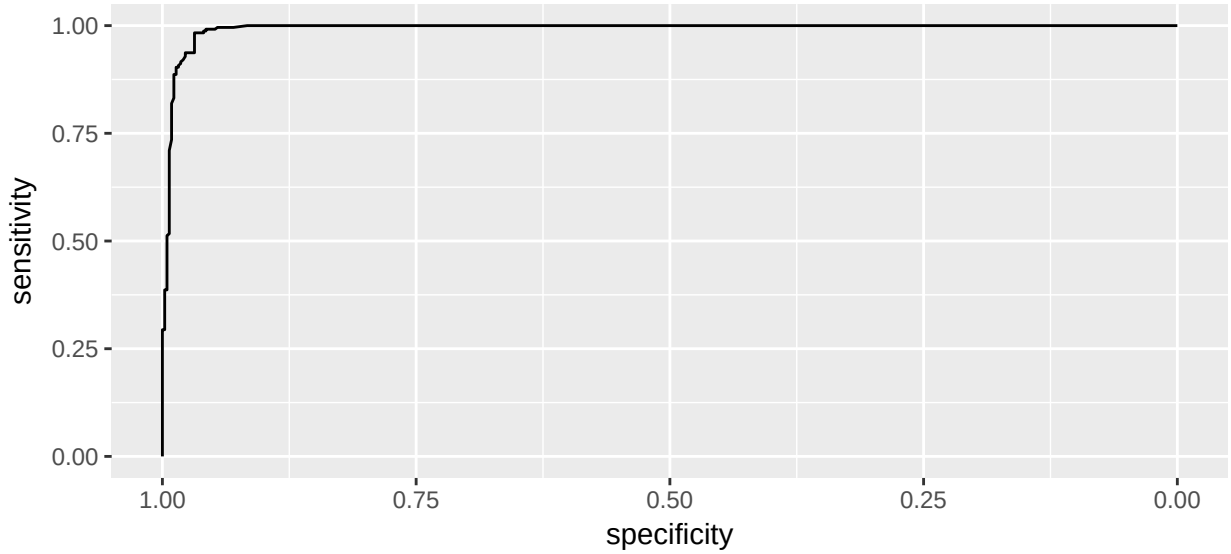
We can think of the True Positive Rate as the probability that a Positive case will be correctly classified as a positive. Similarly a False Positive Rate is the probability that a Negative case will be incorrectly classified as a positive.

I wish to examine the relationship between the False Positive Rate and the True Positive Rate for any decision rule. So what we could do is select a sequence of decision rules and for each calculate the `(FPR, TPR)` pair, and then make a plot where we play connect the dots with the `(FPR, TPR)` pairs.

Of course we don't want to have to do this by hand, so we'll use the package `pROC` to do it for us.

```
# Calculate the ROC information using pROC::roc()
myROC <- roc( wbca$Class,  wbca$phat )
```

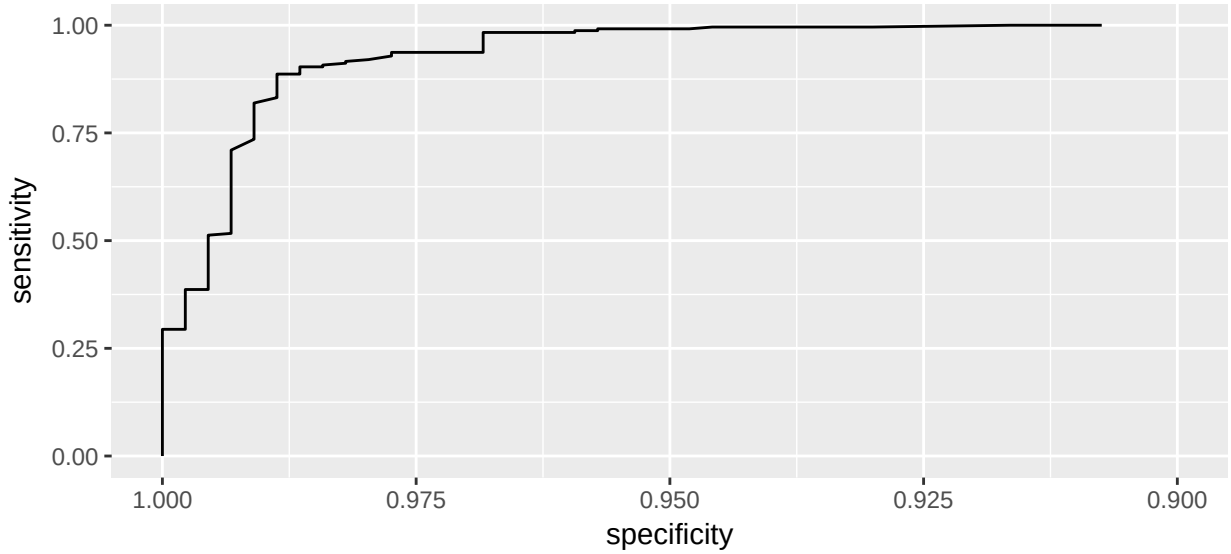
```
# make a nice plot using ggplot2 and pROC::ggroc()
ggroc( myROC )
```



This looks pretty good and in an ideal classifier that makes prefect predictions, this would be a perfect right angle at the bend.

Lets zoom in a little on the high specificity values (i.e. low false positive rates)

```
ggroc( myROC ) + xlim(1, .9)
```



We see that if we want to correctly identify about 99% of maligant tumors, we will have a false positive rate of about 1-0.95 = 0.05. So about 5% of benign tumors would be incorrectly classified as malignant.

It is a little challenging to read the graph to see what the Sensitivity is for a particular value of Specificity. To do this we'll use another function because the authors would prefer you to also estimate the confidence intervals for the quantity. This is a case where bootstrap confidence intervals are quite effective.

```
ci(myROC, of='sp', sensitivities=.99)
```

```
## 95% CI (2000 stratified bootstrap replicates):
```

```
##    se sp.low sp.median sp.high
##  0.99 0.9222    0.9571  0.9797
```

```
ci(myROC, of='se', specificities=.975)
```

```
## 95% CI (2000 stratified bootstrap replicates):
##     sp se.low se.median se.high
##  0.975 0.8824     0.937  0.9958
```

One measure of how far we are from the perfect predictor is the area under the curve. The perfect model would have an area under the curve of 1. For this model the area under the curve is:

```
auc(myROC)
```

```
## Area under the curve: 0.9929
```

```
ci(myROC, of='auc')
```

```
## 95% CI: 0.9878-0.9981 (DeLong)
```

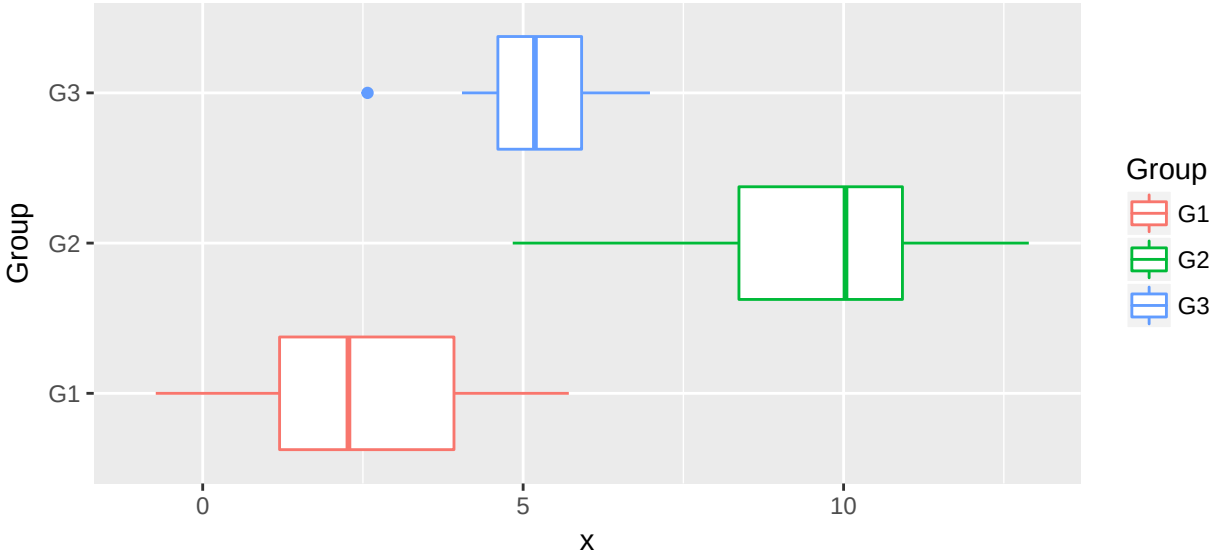
```
ci(myROC, of='auc', method='bootstrap')
```

```
## 95% CI: 0.9875-0.9975 (2000 stratified bootstrap replicates)
```

which seems pretty good and Area Under the Curve (AUC) is often used as a way of comparing the quality of binary classifiers.

4.3 Linear Discriminent Analysis

Discriminent analysis classification techniques assume that we have a categorical (possibly unordered) response y and a continuous covariate predictor x. In many ways we can consider this the inverse problem of a 1-way anova.



Because we are doing ANOVA in reverse, we assume the same relationship between the groups and the continuous covariate, namely that

$$x_{ij} \overset{iid}{\sim} N\left(\mu_i, \sigma^2\right)$$

So to fit normal distributions, we just need to estimate values for the mean of each group $\hat{\mu}_i$ and something for the overal variance, $\hat{\sigma}^2$.

We will use the individual sample means for $\hat{\mu}_i$ and the pooled variance estimator for $\hat{\sigma}^2$. Let k be the number of groups, n_i be the number of samples in group i, and $n = n_1 + \cdots + n_k$ be the total number of observations.

$$\hat{\mu}_i = \bar{x}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} x_{ij}$$

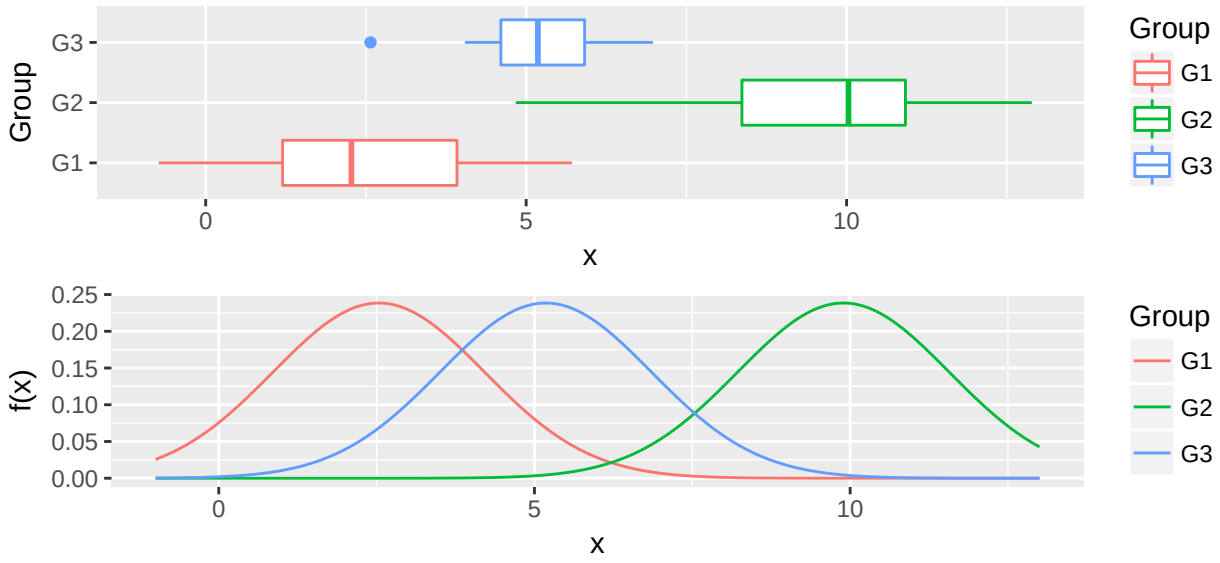
$$\begin{aligned} \hat{\sigma}^2 &= \frac{1}{n-k} \sum_{i=1}^{k} \sum_{j=1}^{n_i} (x_{ij} - \hat{\mu}_i)^2 \\ &= \frac{1}{n-k} \sum_{i=1}^{k} (n_i - 1) s_i^2 \end{aligned}$$

```
str(data)
```

```
## 'data.frame':    90 obs. of  2 variables:
##  $ x    : num  4.41 8.27 6 3.94 10.77 ...
##  $ Group: Factor w/ 3 levels "G1","G2","G3": 1 2 3 1 2 3 1 2 3 1 ...
```

```
n <- nrow(data)
k <- length( levels( data$Group ))

# calculate group level statistics and pooled variance
params <- data %>%
  group_by(Group) %>%
  summarise( xbar = mean(x),
             s2_i = var(x),
             n_i  = n() ) %>%
  mutate( s2 = sum( (n_i -1)*s2_i ) / (n-k) )
```

```
# Calculate the k curves
newdata <- expand.grid( x=seq(-1,13, by=0.01), Group=paste('G', 1:3, sep='') ) %>%
  left_join(params, by='Group') %>%
  mutate( f = dnorm(x, mean=xbar, sd=sqrt(s2) ) )
P2 <- ggplot(newdata, aes(x=x, y=f, color=Group)) +
  geom_line() + xlim(-1, 13) +
  labs( y='f(x)' )

multiplot(P1, P2, ncol=1)
```



Now for any value along the x-axis, say x^*, we predict y^* as whichever group with the largest $\hat{f}_i(x^*)$.

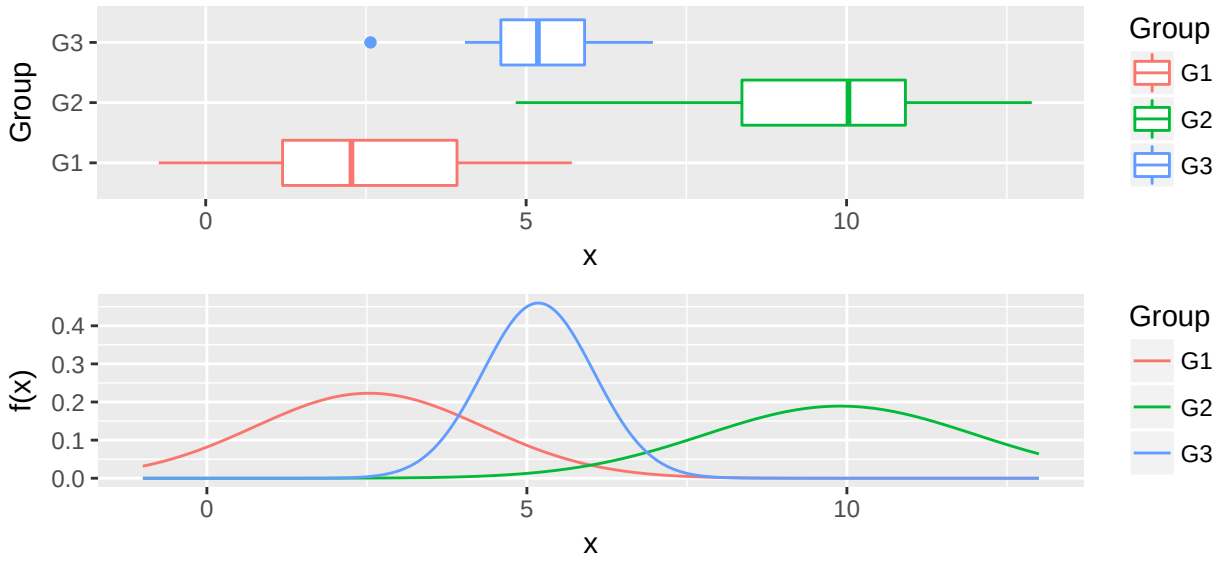
4.4 Quadratic Discriminent Analysis

The only difference between linear discriminent analysis and quadratic, is that we know allow for the groups to have different variance terms.

$$x_{ij} \overset{iid}{\sim} N\left(\mu_i, \sigma_i^2\right)$$

```
n <- nrow(data)
k <- length( levels( data$Group ))

# calculate group level statistics and pooled variance
params <- data %>%
  group_by(Group) %>%
  summarise( xbar = mean(x),
             s2_i = var(x),
             n_i  = n() )
```

```
# Calculate the k curves
newdata <- expand.grid( x=seq(-1,13, by=0.01), Group=paste('G', 1:3, sep='') ) %>%
  left_join(params, by='Group') %>%
  mutate( f = dnorm(x, mean=xbar, sd=sqrt(s2_i) ) )
P2 <- ggplot(newdata, aes(x=x, y=f, color=Group)) +
  geom_line() + xlim(-1, 13) +
  labs( y='f(x)' )

multiplot(P1, P2, ncol=1)
```



The decision to classify a point x^*, y^* as in a particular group is often too stark of a decision, and we would like to assess the certainty of the assignment. We define the *posterior probability* of the point as coming from group i as

$$P(y^* = i|x^*) = \frac{\hat{f}_i(x^*)}{\sum_j \hat{f}_j(x^*)}$$

That is, add up the heights of all the curves at x^* and then use the percent contribution of each to the total as the probability.

4.5 Examples

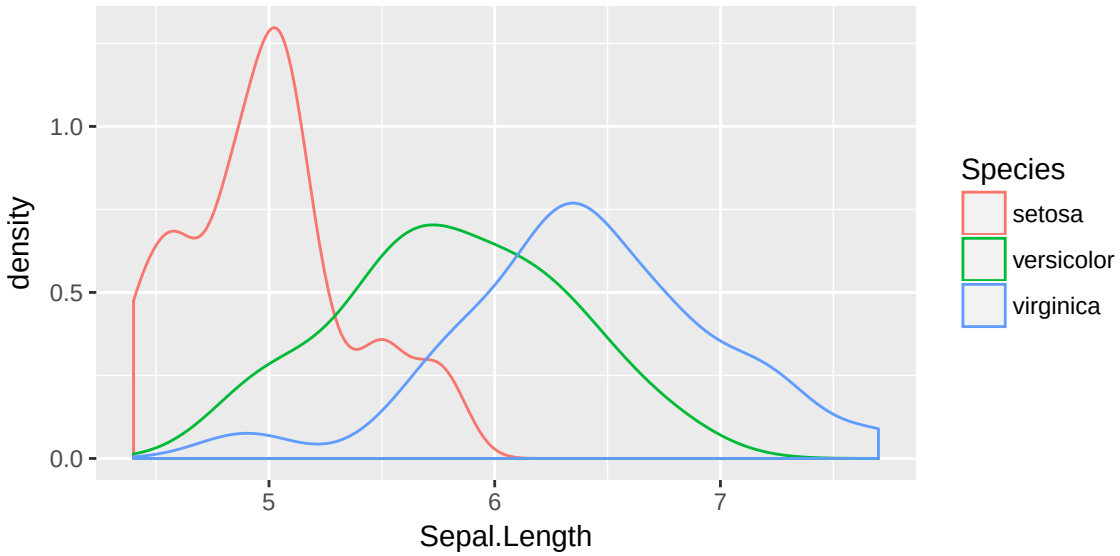
4.5.1 Iris Data

Recall the iris data set is 150 observations that measure leaf and sepal characteristics for three different species of iris. We'll use the leaf characteristics to try to produce a classification rule.

We will first split this into a training data set and a test data set.

```
set.seed( 8675309 )  # so that the same training set is chosen every time...
iris$Obs_ID <- 1:150 # there are a couple of identical rows, we don't like that

# random select 1/2 from each species to be the training set
train <- iris %>% group_by(Species) %>% sample_n( 25 )
test  <- setdiff(iris, train)
```

```
ggplot(train, aes(x=Sepal.Length, color=Species)) +
   geom_density()
```



While these certainly aren't normal and it isn't clear that the equal variance amongst groups is accurate, there is nothing that prevents us from assuming so and just doing LDA.

```
train %>% group_by(Species) %>%
   summarize(xbar = mean(Sepal.Length),
             sd   = sd(Sepal.Length))
```

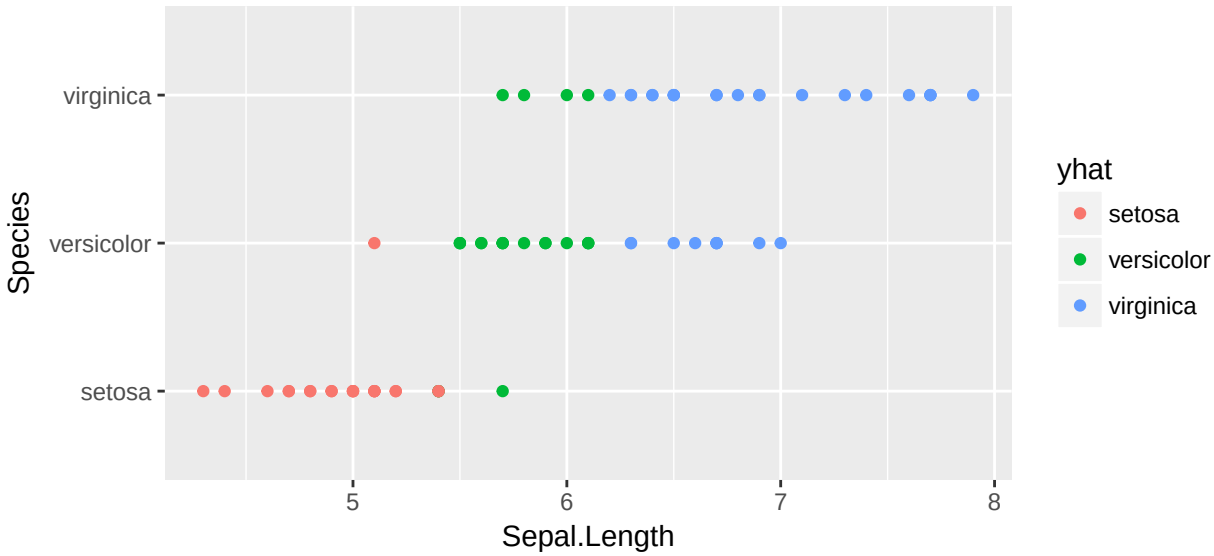
```
## # A tibble: 3 x 3
##      Species  xbar        sd
##       <fctr> <dbl>     <dbl>
## 1     setosa 4.988 0.3756328
## 2 versicolor 5.812 0.5027591
## 3  virginica 6.412 0.5946427
```

We'll predict setosa if x $< (4.988 + 5.812)/2 = 5.4$. We'll predict versicolor if $5.4 <$ x $< (5.812 + 6.412)/2 = 6.1$ We'll predict versicolor if $6.1 <$ x

```
# have R do this
model <- train(Species ~ Sepal.Length, method='lda', data=train)
test$yhat <- predict(model, newdata=test)
table( Truth=test$Species, Prediction=test$yhat )
```

```
##             Prediction
## Truth        setosa versicolor virginica
##   setosa         22          3         0
##   versicolor      1         15         9
##   virginica       0          4        21
```

```
ggplot(test, aes(x=Sepal.Length, y=Species, color=yhat) ) + geom_point()
```



This is a little hard to read, but of the 25 observations that we know are setosa, 3 of them have been misclassified as versicolor. Likewise, of the 25 observations that are virginica, 21 are correctly identified as virginica while 4 are misclassified as versicolor. So out of 75 test cases, we correctly classified 57, and incorrectly classified 17.

```
# Calculate the Misclassification rate
17/75
```

```
## [1] 0.2266667
```

```
test %>%
  summarise( misclassification_rate = mean( Species != yhat ) )
```

```
##   misclassification_rate
## 1              0.2266667
```

Next, we will relax the assumption that the distributions have equal variance.

```
# have R do this
model <- train(Species ~ Sepal.Length, method='qda', data=train)
test$yhat <- predict(model, newdata=test)
table( Truth=test$Species, Prediction=test$yhat )
```

```
##             Prediction
## Truth        setosa versicolor virginica
##   setosa         24          1         0
##   versicolor      1         15         9
##   virginica       0          4        21
```

```
test %>%
  summarise( misclassification_rate = mean( Species != yhat ) )
```

```
##   misclassification_rate
## 1                    0.2
```

This has improved our accuracy as 2 setosa observations that were previously misclassified are now correctly classified.

```
# What about KNN?
model <- train(Species ~ Sepal.Length, method='knn', tuneGrid=data.frame(k=3), data=train)
test$yhat <- predict(model, newdata=test)
```

```
table( Truth=test$Species, Prediction=test$yhat )
```

```
##             Prediction
## Truth        setosa versicolor virginica
##   setosa         19          6         0
##   versicolor      2         13        10
##   virginica       0          5        20
```

```
test %>%
  summarise( misclassification_rate = mean( Species != yhat ) )
```

```
##   misclassification_rate
## 1              0.3066667
```

4.5.2 Detecting Blood Doping

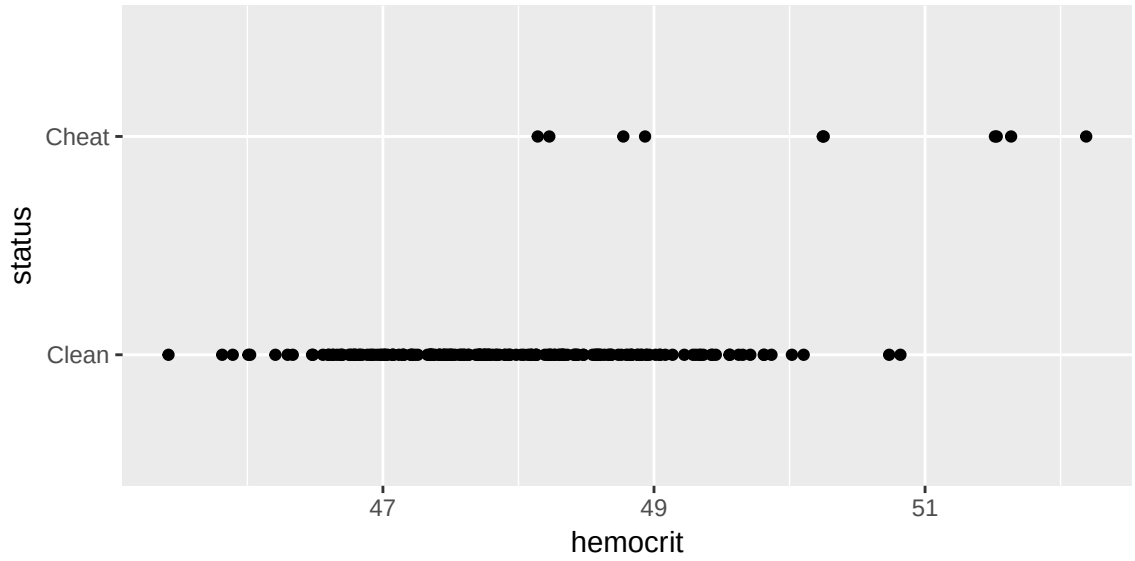
```
library(devtools)
install_github('dereksonderegger/dsData')
```

```
## Skipping install of 'dsData' from a github remote, the SHA1 (208be54a) has not changed since last install.
##   Use `force = TRUE` to force installation
```

```
library(dsData)
```

We now consider a case where the number of observations is not the same between groups. Here we consider the case where we are interested in using hemocrit levels to detect if a cyclist is cheating.

```
data('Hemocrit', package='dsData')
ggplot(Hemocrit, aes(x=hemocrit, y=status)) +
  geom_point()
```



What if I just naively assume that all professional cyclists are clean? How accurate is this prediction scheme?

```
mean(  Hemocrit$status == 'Clean' )
```

```
## [1] 0.95
```

In this case, I am pretty accurate because we correctly classify 95% of the cases! Clearly we should be more intelligent. Lets use the LDA to fit a model that uses hemocrit.

```
model <- train( status ~ hemocrit, method='lda', data=Hemocrit)
Hemocrit$yhat <- predict(model)
table( Truth=Hemocrit$status, Predicted=Hemocrit$yhat)
```

```
##        Predicted
## Truth   Clean Cheat
##   Clean   188     2
##   Cheat     6     4
```

So this method basically looks to see if the hemocrit level is greater than

```
Hemocrit %>% group_by(status) %>%
  summarise(xbar=mean(hemocrit))
```

```
## # A tibble: 2 x 2
##   status     xbar
##   <fctr>    <dbl>
## 1  Clean 47.91675
## 2  Cheat 50.14432
```

```
(47.917 + 50.144)/2
```

```
## [1] 49.0305
```

and calls them a cheater. Can we choose something a bit more clever? The `predict` function has an option where it returns the class probabilities. By default, it chooses the category with the highest probability (for two classes than means whichever is greater than 0.50). We can create a different rule that labels somebody a cheater only if the posterior probability is greater than 0.8 or whatever.

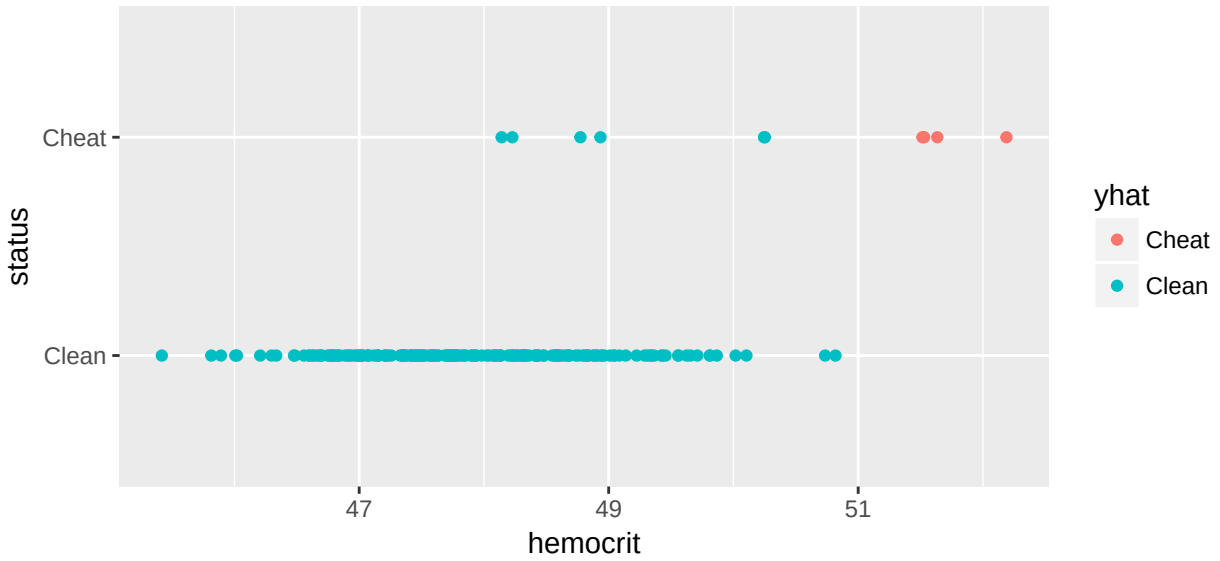
```
pred <- predict(model, type='prob')
Hemocrit <- Hemocrit %>%
  mutate( phat = pred$Cheat,
          yhat = ifelse(phat <= .8, 'Clean', 'Cheat' ))
table( Truth=Hemocrit$status, Predicted=Hemocrit$yhat)
```

```
##        Predicted
## Truth   Cheat Clean
##   Clean     0   190
##   Cheat     4     6
```

```
mean(  Hemocrit$status == Hemocrit$yhat )
```

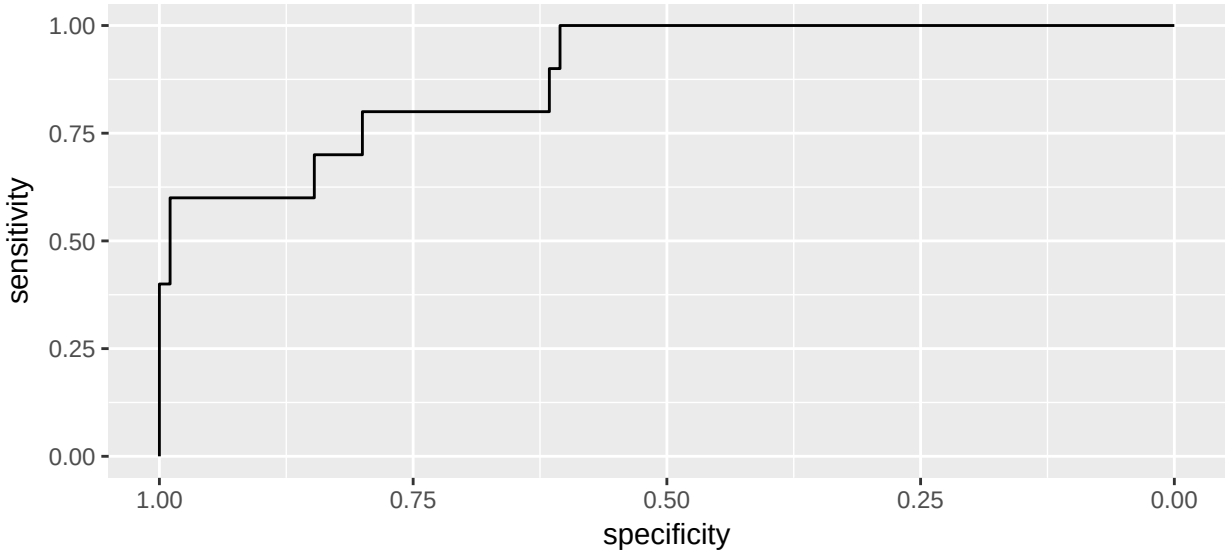
```
## [1] 0.97
```

```
ggplot(Hemocrit, aes(x=hemocrit, y=status, color=yhat)) +
  geom_point()
```



Great, now we have no false-positives, but a number of folks are getting away with cheating. But what if we back that up, how many false positives do we get… What we want is a graph that compares my false-positive numbers to the true-positives.

```
P3 <- pROC::roc(Hemocrit$status, Hemocrit$phat)
pROC::ggroc(P3)
```



4.5.3 2-d Example

I need an example where we have a two-dimensional case where we can display the normal density curves as contour lines and see the boundary line curve in the QDA case and not in the LDA. Also getting to see that variance in the 2-d case is spread and correlation.

4.6 Exercises

1. ISL Chapter 4, problem 4: When the number of features p is large, there tends to be a deterioration in the performance of KNN and other local approaches that perform prediction using only observations

that are near the test observation for which a prediction must be made. This phenomenon is known as *the curse of dimensionality*, and it ties into the fact that non-parametric approaches often perform poorly when p is large. We will now investigate this curse.

a) Suppose that we have a set of observations, each with measurements on $p = 1$ feature, X. We assume that X is uniformly (evenly) distributed on $[0, 1]$. Associated with each observation is a response value. Suppose that we wish to predict a test observation's response using only observations that are within 10% of the range of X closest to that test observation. For instance, in order to predict the response for a test observation with $X = 0.6$, we will use observations in the range $[0.55, 0.65]$. On average, what fraction of the available observations will we use to make the prediction?

b) Now suppose that we have a set of observations, each with measurements on $p = 2$ features, X_1 and X_2. We assume that (X_1, X_2) are uniformly distributed on $[0, 1]$»$[0, 1]$. We wish to predict a test observation's response using only observations that are within 10% of the range of X_1 and within 10% of the range of X_2 closest to that test observation. For instance, in order to predict the response for a test observation with $X_1 = 0.6$ and $X_2 = 0.35$, we will use observations in the range $[0.55, 0.65]$ for X_1 and in the range $[0.3, 0.4]$ for X_2. On average, what fraction of the available observations will we use to make the prediction?

c) Now suppose that we have a set of observations on $p = 100$ features. Again the observations are uniformly distributed on each feature, and again each feature ranges in value from 0 to 1. We wish to predict a test observation's response using observations within the 10% of each feature's range that is closest to that test observation. What fraction of the available observations will we use to make the prediction?

d) Using your answers to parts (a)–(c), argue that a drawback of KNN when p is large is that there are very few training observations "near" any given test observation.

e) Now suppose that we wish to make a prediction for a test observation by creating a p-dimensional hypercube centered around the test observation that contains, on average, 10% of the training observations. For p = 1, 2, and 100, what is the length of each side of the hypercube? Comment on your answer. *Hint: Use the equation from part (c).*

2. ISL Chapter 4, problem 5: We now examine the differences between LDA and QDA.

a) If the Bayes decision boundary is linear, do we expect LDA or QDA to perform better on the training set? On the test set?
b) If the Bayes decision boundary is non-linear, do we expect LDA or QDA to perform better on the training set? On the test set?
c) In general, as the sample size n increases, do we expect the test prediction accuracy of QDA relative to LDA to improve, decline, or be unchanged? Why?
d) True or False: Even if the Bayes decision boundary for a given problem is linear, we will probably achieve a superior test error rate using QDA rather than LDA because QDA is flexible enough to model a linear decision boundary. Justify your answer.

3. ISL Chapter 4, problem 6: Suppose we collect data for a group of students in a statistics class with variables X_1 = hours studied, X_2 = undergrad GPA, and Y = receive an A. We fit a logistic regression and produce estimated coefficient, $\hat{\beta}_0 = -6$, $\hat{\beta}_1 = 0.05$, $\hat{\beta}_2 = 1$.

a) Estimate the probability that a student who studies for 40 hours and has an undergrad GPA of 3.5 gets an A in the class.
b) How many hours would the student in part (a) need to study to have a 50% chance of getting an A in the class?

4. ISL Chapter 4, problem 8: Suppose that we take a data set, divide it into equally-sized training and test sets, and then try out two different classification procedures. First we use logistic regression and get an error rate of 20% on the training data and 30% on the test data. Next we use 1-nearest neighbors (i.e. K = 1) and get an average error rate (averaged over both test and training data sets) of 18%.

Based on these results, which method should we prefer to use for classification of new observations? Why?

5. ISL Chapter 4, problem 11: In this problem, you will develop a model to predict whether a given car gets high or low gas mileage based on the `Auto` data set that is included in the package `ISLR`.
   a) Create a binary variable, `mpg01`, that contains a 1 if mpg contains a value above its median, and a 0 if mpg contains a value below its median. You can compute the median using the `median()` function. Add this column to the `Auto` data set. *Hint: You also need to make sure this is a factor or else your modeling functions will complain. Use the `factor()` command to do so.*
   b) Explore the data graphically in order to investigate the association between mpg01 and the other features. Which of the other features seem most likely to be useful in predicting mpg01? Scatterplots and boxplots may be useful tools to answer this question. Describe your findings.
   c) Split the data into equally sized training and test sets.
   d) Perform LDA on the training data in order to predict mpg01 using the variables that seemed most associated with `mpg01` in (b). What is the test error of the model obtained?
   e) Perform QDA on the training data in order to predict mpg01 using the variables that seemed most associated with mpg01 in (b). What is the test error of the model obtained?
   f) Perform logistic regression on the training data in order to predict `mpg01` using the variables that seemed most associated with `mpg01` in (b). What is the test error of the model obtained?
   g) Perform KNN on the training data, with several values of K, in order to predict `mpg01`. Use only the variables that seemed most associated with `mpg01` in (b). What test errors do you obtain? Which value of K seems to perform the best on this data set?
6. ISL Chapter 4, problem 13: Using the `ISLR::Boston` data set, fit classification models in order to predict whether a given suburb has a crime rate above or below the median. Explore logistic regression, LDA, and KNN models using various subsets of the predictors. Describe your findings.

Chapter 5

Resampling Methods

Resampling methods are an important tool in modern statistics. They are applicable in a wide range of situations and require minimal theoretical advances to be useful in new situations. However, these methods require a large amount of computing effort and care must be taken to avoid excessive calculation.

The main idea for these methods is that we will repeatedly draw samples from the training set and fit a model on each sample. From each model we will extract a statistic of interest and then examine the distribution of the statistic across the simulated samples.

We will primarily discuss *cross-validation* and *bootstrapping* in this chapter. I think of cross-validation as a model selection and assessment tool while bootstrap is an inferential tool for creating confidence intervals.

5.1 Cross-validation

```
library(caret)        # train/predict interface to gam
library(gam)          # for my spline stuff
library(boot)
library(car)
library(STA578)       # For multiplot()
library(ggplot2)
library(dplyr)
library(stringr)      # string manipulation stuff
```

We are primarily interested in considering models of the form

$$y = f(x) + \epsilon$$

and we wish to estimate f with $\hat{f}$ and we also wish to understand $Var(\epsilon)$. We decided a good approach would be to split our observed data into test/training sets and then using the training set to produce $\hat{f}$ and use it to predict values in the test set $\hat{y_i} = \hat{f}(x_i)$ and then use

$$MSE = \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} (y_i - \hat{y}_i)^2$$

as an estimate for $Var(\epsilon)$.

Once we have estimated the function $f()$ with some method, we wish to evaluate how well the model predicts the observed data, and how well it is likely to predict new data. We have looked at the bias/variance

relationship of the prediction error for a new observations, (x_0, y_0) as

$$E\left[(y_0 - \hat{f}(x_0))^2\right] = Var(\hat{f}) + \left[Bias(\hat{f})\right]^2 + Var(\epsilon)$$

where

- $Var(\hat{f})$ is how much our estimated function will vary if we had a completely new set of data.
- $Bias(\hat{f})$ is how much our estimated function differs from the true f.

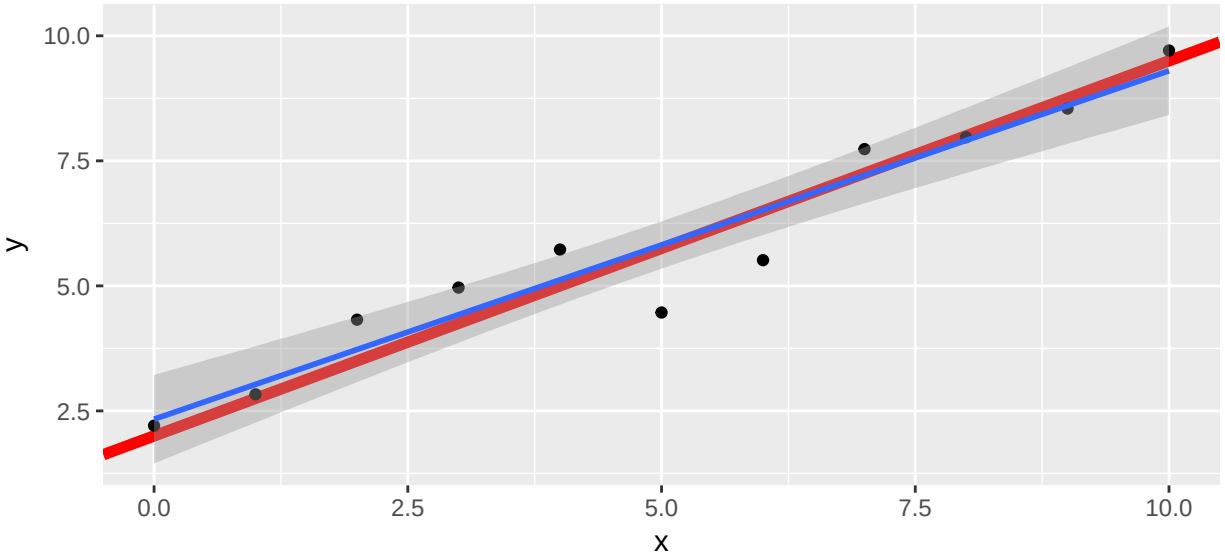
Notice that all the terms on my right hand side of this equation are positive so using the test set MSE will tend to overestimate $Var(\epsilon)$. In other words, test set MSE is a *biased* estimate of $Var(\epsilon)$. For a particular set of data the test MSE is calculated using only a single instance of $\hat{f}$ and so averaging across test observations won't fix the fact that $\hat{f} \neq f$. Only by repeated fitting $\hat{f}$ on *different* sets of data could the bias term be knocked out of the test MSE. However as our sample size increase, this overestimation will decrease because the bias will decrease because $\hat{f}$ will be closer to f and the variance of f will also be less.

Lets do a simulation study to show this is true. We will generate data from a simple linear regression model, split the data equally into training and testing sets, and then use the test MSE as an estimate of $Var(\epsilon)$. As we look at test set MSE as an estimator for the $Var(\epsilon)$, should look for what values of n tend to have an unbiased estimate of $\sigma = 1$ and also have the smallest variance. Confusingly we are interested in the variance of the variance estimator, but that is why this is a graduate course.

```
# what will a simulated set of data look like, along with a simple regression
# Red line = True f(x)
n <- 10
sigma <- 1  # variance is also 1
data <- data.frame( x= seq(0,10) ) %>%
  mutate( y = 2 + .75*x + rnorm(n, sd=sigma) )
```

```
## Warning in 2 + 0.75 * x + rnorm(n, sd = sigma): longer object length is not
## a multiple of shorter object length
```

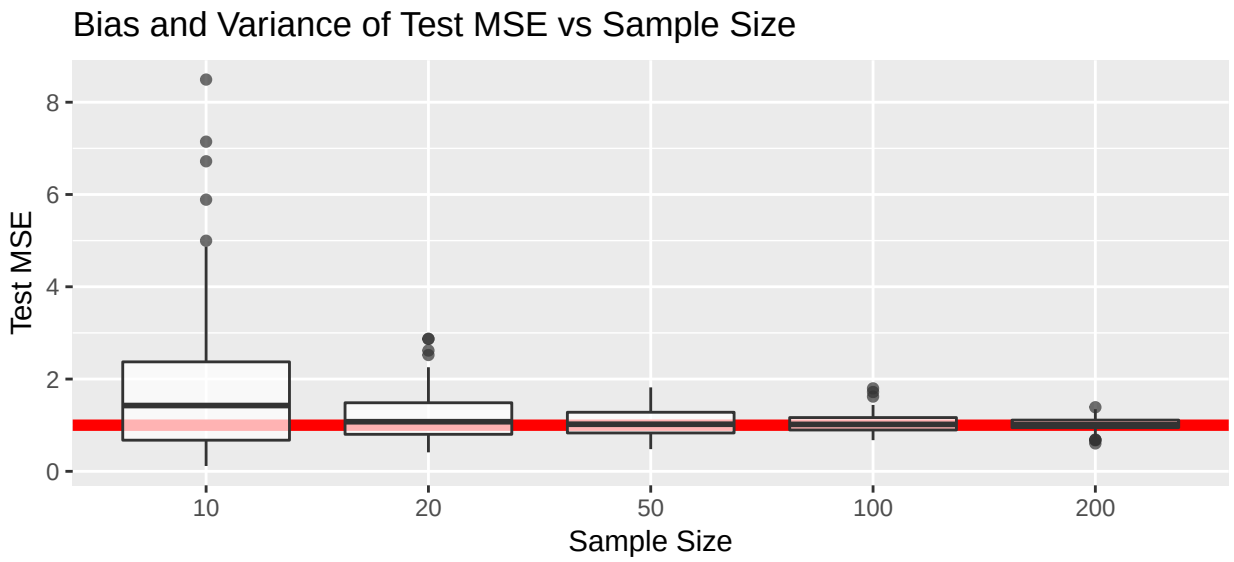
```
ggplot(data, aes(x=x, y=y) ) +
  geom_abline(slope=.75, intercept = 2, color='red', size=2) +
  geom_point() +
  geom_smooth(method='lm')
```



Now to do these simulations.

```
M <- 100  # do 100 simulations for each sample size n
results <- NULL
for( n in c(10, 20, 50, 100,  200)){
  for(i in 1:M){
    data <- data.frame( x= seq(0,10, length.out = n) ) %>%
      mutate( y = 2 + .75*x + rnorm(n, sd=sigma) )
    train <- data %>% sample_frac(.5)
    test  <- setdiff(data, train)
    model <- train( y ~ x, data=train, method='lm')
    test$yhat  <- predict(model, newdata=test)
    output <- data.frame(n=n, rep=i, MSE = mean( (test$y - test$yhat)^2 ))
    results <- rbind(results, output)
  }
}
save(results, file="Simulations/OverEstimation.RData")
```

```
# plot the results
load('Simulations/OverEstimation.RData')
ggplot(results, aes(x=factor(n), y=MSE)) +
  geom_abline(intercept = 1, slope=0, color='red', size=2) +
  geom_boxplot(alpha=.7) +
  labs(y='Test MSE', x='Sample Size', title='Bias and Variance of Test MSE vs Sample Size')
```



As we discussed earlier from a theoretical perspective, test MSE tends to overestimate $Var(\epsilon)$ but does better with a larger sample size. Similarly the variance of test MSE also tends to get smaller with larger sample sizes.

We now turn our focus to examining several different ways we could use the train/test paradigm to estimate $Var(\epsilon)$. For each method, we will again generate data from a simple regression model and then fit a linear model and therefore we shouldn't have any mis-specification error and we can focus on which procedure produces the least biased and minimum variance estimate of $Var(\epsilon)$. For each simulation we will create $M = 100$ datasets and examine the resulting MSE values.

```
# Simulation parameters
M <- 100      # number of idependent simulations per method
n <- 50       # Sample size per simulation
sigma <- 1    # var(epsilon) = stddev(epsilon)
```

```
results <- NULL
```

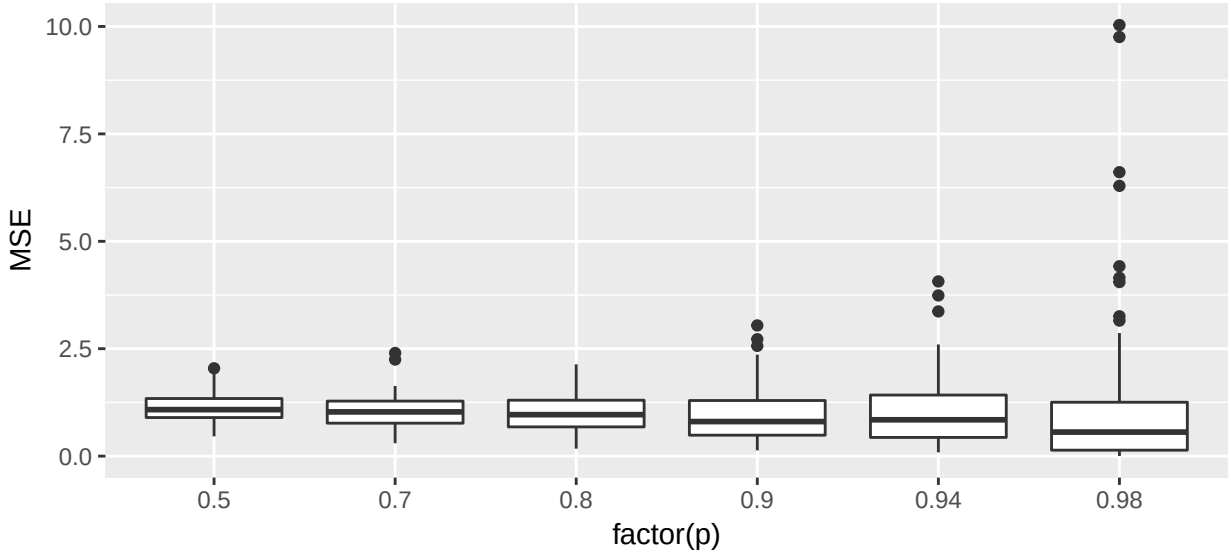
5.1.1 Validation Sets Approach

This is the approach that we just pursued. In the validation sets approach, we: 1. Randomly split the data into training and test sets, where the proportion p is assigned to the training set. 2. Fit a model to the training set 3. Use the model to predict values in the test set 4. MSE = mean squared error of values in the test set:

$$MSE = \frac{1}{n_{test}} \sum (y_i - \hat{y}_i)^2$$

```
ValidationSets_results <- NULL
M <- 100
for(j in 1:M){
  for( p in c(.5, .7, .8, .9, .94, .98) ){
    data <- data.frame( x= seq(0,10, length.out = n) ) %>%
        mutate( y = 2 + .75*x + rnorm(n, sd=sigma) )
    train <- data %>% sample_frac(p)
    test  <- setdiff(data, train)
    model <- train( y ~ x, data=train, method='lm')
    test$yhat  <- predict(model, newdata=test)
    output <- data.frame(p=p, rep=j, MSE = mean( (test$y - test$yhat)^2 ))
    ValidationSets_results <- rbind(ValidationSets_results, output)
  }
}
save(ValidationSets_results, file='Simulations/LinearModel_ValidationSets.RData')
```

```
load('Simulations/LinearModel_ValidationSets.RData')
ggplot(ValidationSets_results, aes(x=factor(p), y=MSE)) + geom_boxplot()
```




```
ValidationSets_results %>%
  group_by(p) %>%
  summarise(mean_MSE = mean(MSE))
```

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

```
##   <dbl>     <dbl>
## 1  0.50 1.1148614
## 2  0.70 1.0405579
## 3  0.80 1.0196439
## 4  0.90 0.9923938
## 5  0.94 1.0421772
## 6  0.98 1.1326722
```

When we train our model on more data, we see smaller MSE values to a point, but in the extreme (where we train on 98% and test on 2%, where in this case it is train on 49 observations and test on 1) we have much higher variability. If we were take the mean of all $M = 100$ simulations, we would see that the average MSE is near 1 for each of these proportions. So the best looking options are holding out 20 to 50%.

5.1.2 Leave one out Cross Validation (LOOCV).

Instead of randomly selecting one observation to be the test observation, LOOCV has each observation take a turn at being left out, the we average together all the predicted squared errors.

```
LOOCV_results <- NULL
M <- 100
for(j in 1:M){
  data <- data.frame( x= seq(0,10, length.out = n) ) %>%
      mutate( y = 2 + .75*x + rnorm(n, sd=sigma) )
  for( i in 1:n  ){
    train <- data[ -i, ]
    test  <- data[  i, ]
    model <- train( y ~ x, data=train, method='lm')
    data[i,'yhat']  <- predict(model, newdata=test)
  }
  output <- data.frame(rep=j, MSE = mean( (data$y - data$yhat)^2 ))
  LOOCV_results <- rbind(LOOCV_results, output)
}
save(ValidationSets_results, file='Simulations/LinearModel_LOOCV.RData')
```

```
# load('Simulations/LinearModel_LOOCV.RData')
# Total_Results <- rbind(
#   ValidationSets_results %>% mutate(method=str_c('VS_',p)) %>% dplyr::select(MSE,method),
#   LOOCV_results %>% mutate(method='LOOCV') %>% dplyr::select(MSE, method)) %>%
#   filter(is.element(method, c('VS_0.5', 'VS_0.7', 'VS_0.8')))
# ggplot(Total_Results, aes(x=method, y=MSE)) + geom_boxplot()
```

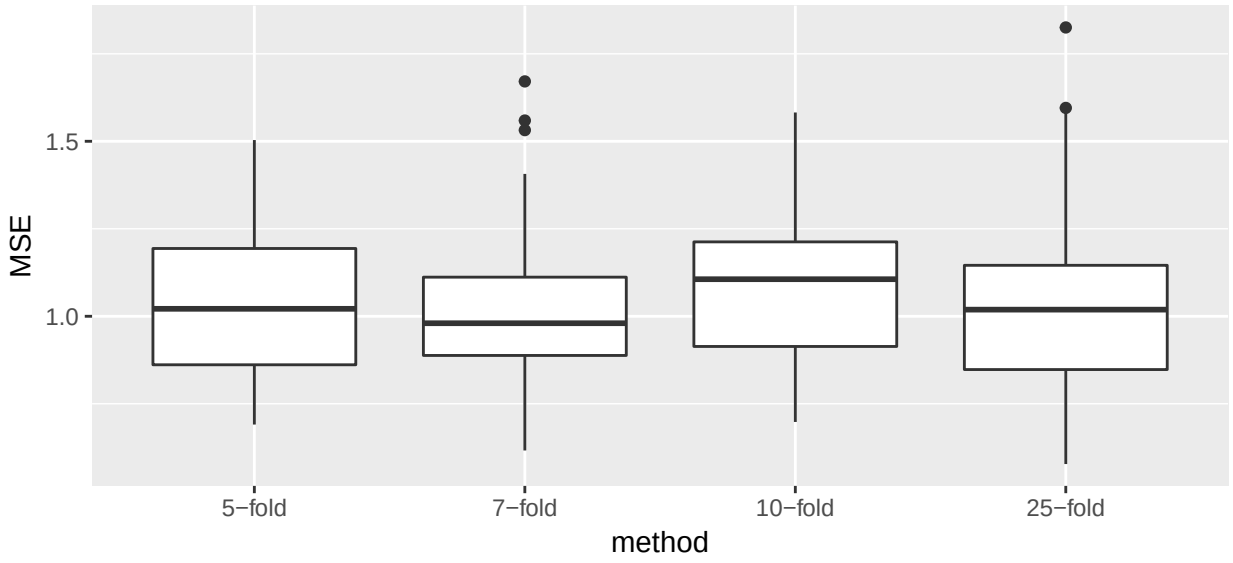
This was extremely painful to perform because there were so many model fits. If we had a larger n it would be computationally prohibative. In general, we ignore LOOCV because of the computational intensity. By taking each observation out in turn, we reduced the high variability that we saw in the validation sets method with $p = 0.98$.

5.1.3 K-fold cross validation

A computational compromise between LOOCV and validation sets is K-fold cross validation. Here we randomly assign each observation to one of K groups. In turn, we remove a group, fit the model on the rest of the data, then make predict for the removed group. Finally the MSE is the average prediction error for all observations.

```
KfoldCV_results <- NULL
M <- 100
for(j in 1:M){
  for( K in c(5, 7, 10, 25) ){
    data <- data.frame( x= seq(0,10, length.out = n) ) %>%
        mutate( y = 2 + .75*x + rnorm(n, sd=sigma) ) %>%
        mutate(fold = sample( rep(1:K, times=ceiling(n/K))[1:n] ) )
    for( k in 1:K  ){
      index <- which( data$fold == k )
      train <- data[ -index, ]
      test  <- data[  index, ]
      model <- train( y ~ x, data=train, method='lm')
      data[index,'yhat']  <- predict(model, newdata=test)
    }
    output <- data.frame(R=1, K=K, MSE = mean( (data$y - data$yhat)^2 ))
    KfoldCV_results <- rbind(KfoldCV_results, output)
  }
}
save(KfoldCV_results, file='Simulations/LinearModel_KfoldCV.RData')
```

```
load('Simulations/LinearModel_KfoldCV.RData')
Total_Results <- NULL
Total_Results <- rbind(
  Total_Results,
  KfoldCV_results %>% mutate(method=str_c(K,'-fold')) %>% dplyr::select(MSE, method) %>%
    mutate(method=factor(method, levels=c('5-fold','7-fold','10-fold','25-fold'))))

ggplot(Total_Results, aes(x=method, y=MSE)) + geom_boxplot()
```



This looks really good for K-fold cross validation. By still having quite a lot of data in the training set, the estimates have relatively low bias (undetectable really for $n = 50$), and the variability of the estimator is much smaller due to averaging across several folds. However, we do see that as the number of folds increases, and thus the number of elements in each test set gets small, the variance increases.

5.1.4 Repeated K-fold cross validation

By averaging across folds we reduce variability, but we still want the size of the test group to be large enough. So we could *repeatedly* perform K-fold cross validation and calculate the MSE by averaging across all the repeated folds.

```
R_x_KfoldCV_results <- NULL
M <- 100
for(j in 1:M){
  for( R in c(2,4,6,8) ){
    for( K in c(5, 7, 10, 25) ){
      data <- data.frame( x= seq(0,10, length.out = n) ) %>%
          mutate( y = 2 + .75*x + rnorm(n, sd=sigma) ) %>%
          mutate(fold = sample( rep(1:K, times=ceiling(n/K))[1:n] ) )
      Sim_J_Result <- NULL
      for( r in 1:R ){
        for( k in 1:K  ){
          index <- which( data$fold == k )
          train <- data[ -index, ]
          test  <- data[  index, ]
          model <- train( y ~ x, data=train, method='lm')
          test$yhat  <- predict(model, newdata=test)
          Sim_J_Result <- data.frame( MSE = mean( (test$y - test$yhat)^2 ) )
        }
      }
      output <- data.frame(R=R, K=K, MSE = mean( Sim_J_Result$MSE ))
      R_x_KfoldCV_results <- rbind(R_x_KfoldCV_results, output)
    }
  }
}
save(R_x_KfoldCV_results, 'Simulations/LinearModel_R_x_KfoldCV.RData')
```

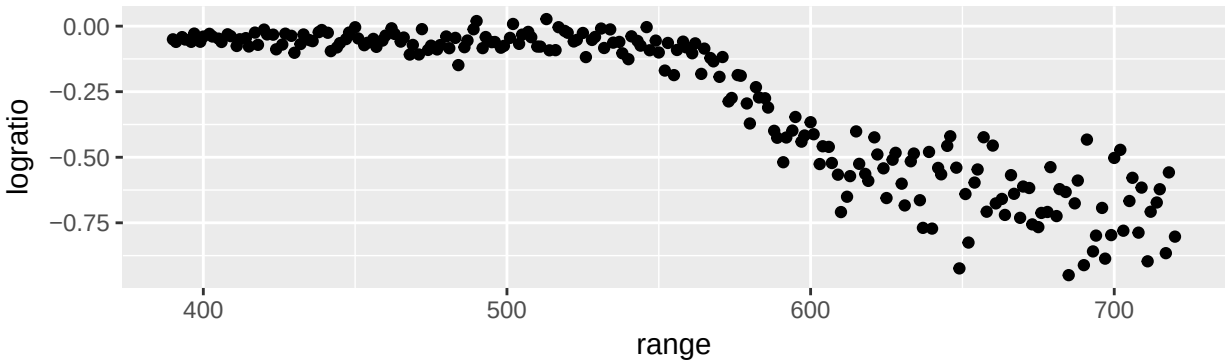
```
# load('Simulations/LinearModel_R_x_KfoldCV.RData')
# All_R_x_KfoldCV_results <- cbind(KfoldCV_results, R_x_KfoldCV_results)
# ggplot(All_R_x_KfoldCV_results, aes(x=method, y=MSE)) + geom_boxplot() +
#   facet_grid( .~R)
```

Repeated was ok??? Need these actual results.

5.1.5 Using cross validation to select a tuning parameter

First we'll load some data to work with from the library 'SemiPar'

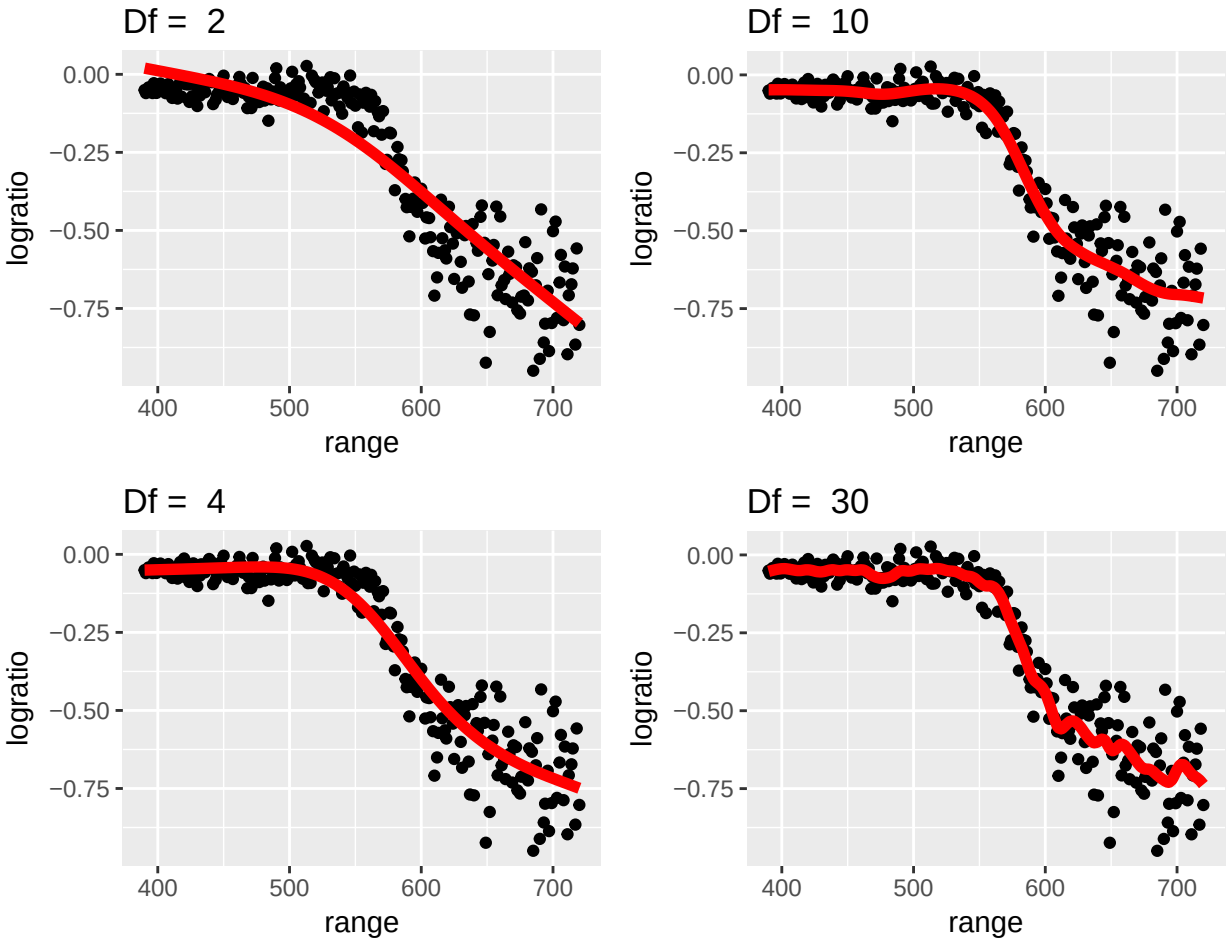
```
data('lidar', package='SemiPar')
ggplot(lidar, aes(x=range, y=logratio)) +
  geom_point()
```



We'll fit this data using a Regression Spline (see chapter 7), but all we need for now is that there is a flexibility parameter that is related to how smooth the function is.

```
df <- c(2,4,10,30)
P <- list()
for( i in 1:length(df) ){
  model <- train(logratio ~ range, data=lidar,
                 method='gamSpline', tuneGrid=data.frame(df=df[i]) )
  lidar$fit <- predict(model)

  P[[i]] <- ggplot(lidar, aes(x=range)) +
    geom_point(aes(y=logratio)) +
    geom_line(aes(y=fit), color='red', size=2) +
    ggtitle(paste('Df = ', df[i]))
}
```

```
STA578::multiplot(P[[1]], P[[2]], P[[3]], P[[4]], ncol=2)
```



Looking at these graphs, it seems apparent that having `df=6` to `8` is approximately correct. Lets see what model is best using cross validation. Furthermore, we will use the package `caret` to do this instead of coding all of this by hand.

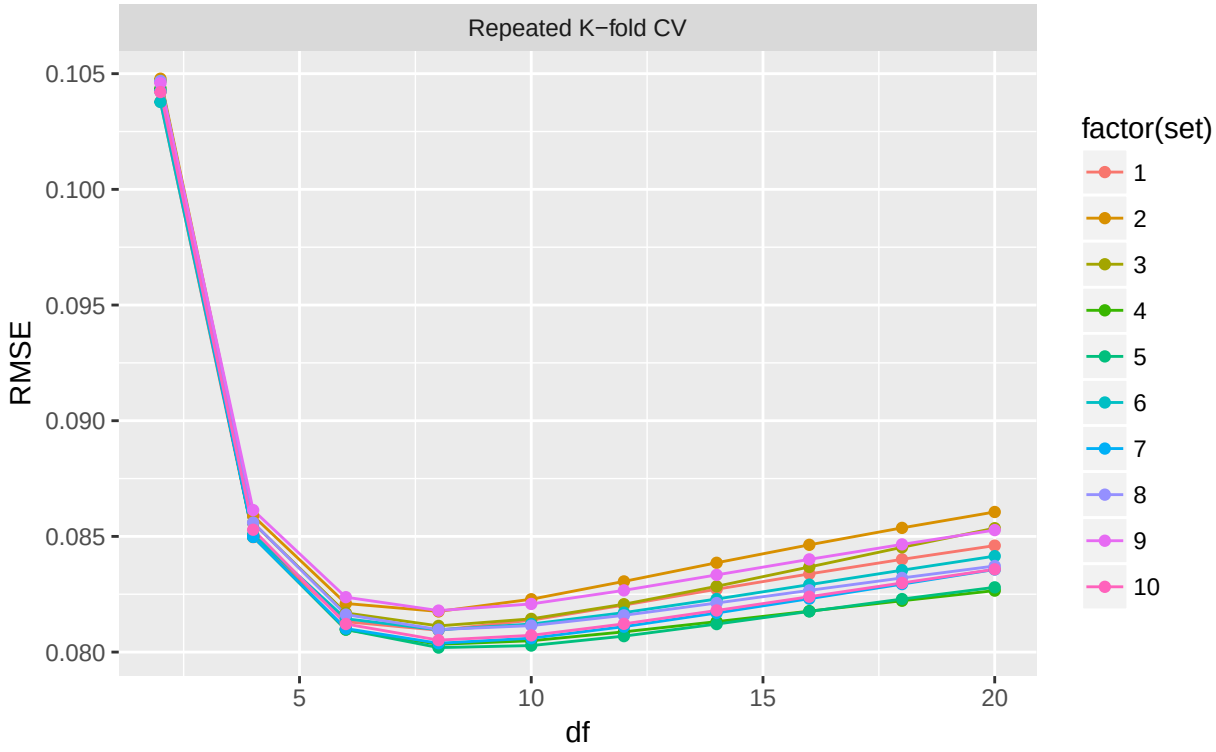
The primary way to interact with `caret` is through the `train()` function and we notice that until now, we've always passed a single value into the `tuneGrid` parameter. By passing multiple values, we create a set of tuning parameters to select from using cross validation. We will control the manner in which we perform the cross validation using the `trControl` parameter.

The output of the `train` function has two important elements, `results`, which is the RMSE for each row in the `tuneGrid` and `bestTune` which gives the row with the smallest RMSE.

```
# Repeated 4x5-fold Cross Validation
ctrl <- trainControl( method='repeatedcv', number=5, repeats=4 )
grid <- data.frame( df = seq(2, 20, by=2 ) )

rkfold_output <- NULL
for( s in 1:10 ){
  model <- train(logratio ~ range, data=lidar, method='gamSpline',
                 trControl = ctrl, tuneGrid=grid )
  results <- model$results %>%
    dplyr::select(df, RMSE) %>%
    mutate( method='Repeated K-fold CV', set=s )
  rkfold_output <- rbind( rkfold_output, results )
}
```

Finally we can make a graph showing the output of each.

```
ggplot(rkfold_output, aes(x=df, y=RMSE, color=factor(set))) +
  geom_line() + geom_point() +
  facet_wrap( ~ method )
```

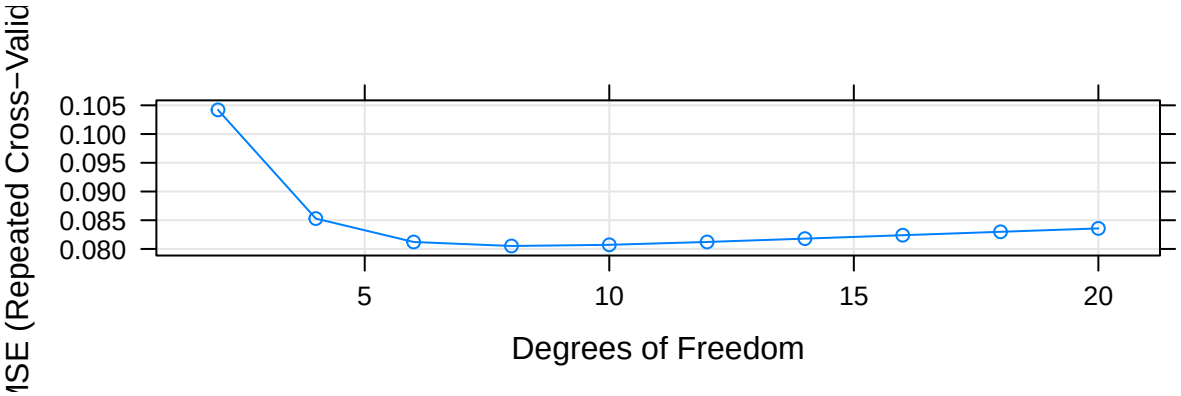


Clearly each of the `s=10` runs chose `df=8` to be the best choice and so we didn't need to run all 10.

```
model$results
```

```
##    df       RMSE  Rsquared      RMSESD RsquaredSD
## 1   2 0.10420558 0.8685406 0.008270251 0.02412875
## 2   4 0.08528731 0.9118554 0.007659893 0.01570068
## 3   6 0.08121312 0.9198476 0.008516047 0.01436658
## 4   8 0.08051589 0.9210965 0.009096070 0.01513959
## 5  10 0.08072231 0.9206112 0.009541862 0.01623587
## 6  12 0.08121948 0.9196103 0.009881607 0.01721261
## 7  14 0.08179807 0.9184968 0.010107235 0.01795221
## 8  16 0.08239242 0.9173832 0.010247375 0.01850647
## 9  18 0.08298687 0.9162848 0.010338705 0.01895523
## 10 20 0.08357659 0.9151983 0.010409663 0.01935861
```

```
plot(model)
```

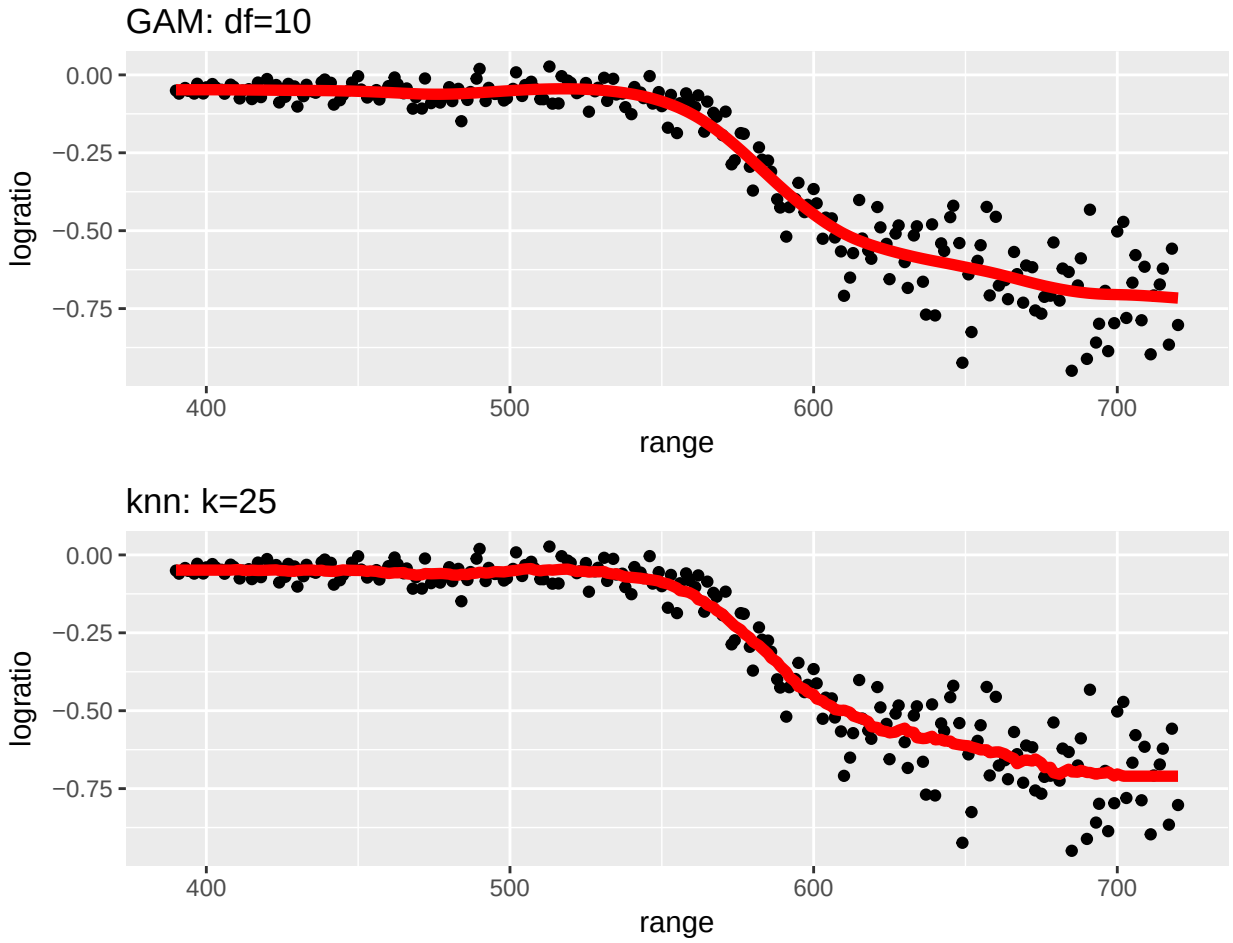


5.1.6 Comparing two analysis techniques

Finally we will compare different analysis techniques again using cross validation. We will fit the 'lidar' data using either the `gamSplines` or using `knn`.

```
# Repeated 4x5-fold Cross Validation
ctrl <- trainControl( method='repeatedcv', repeats=4, number=5)

set.seed(8675309) # same fold assignment!
Spline <- train(logratio ~ range, data=lidar, trControl = ctrl,
                method='gamSpline', tuneGrid=data.frame(df=seq(2,20,by=2)) )

set.seed(8675309) # same fold assignments!
knn    <- train(logratio ~ range, data=lidar, trControl = ctrl,
                method='knn', tuneGrid=data.frame(k=seq(5,80,by=5)))
```

```
lidar$yhat <- predict(Spline)
P1 <- ggplot(lidar, aes(x=range, y=logratio)) +
  geom_point( aes(y=logratio) ) +
  geom_line( aes(y=yhat), color='red', size=2 ) +
  labs( title = str_c('GAM: df=', Spline$bestTune$df) )

lidar$yhat <- predict(knn)
P2 <- ggplot(lidar, aes(x=range, y=logratio)) +
  geom_point( aes(y=logratio) ) +
  geom_line( aes(y=yhat), color='red', size=2 ) +
  labs( title = str_c('knn: k=', knn$bestTune$k) )

STA578::multiplot(P1, P2)
```



Both of these techniques fit the observed data quite well, though I worry that some of the wiggliness in KNN for large range values is overfitting the noise.

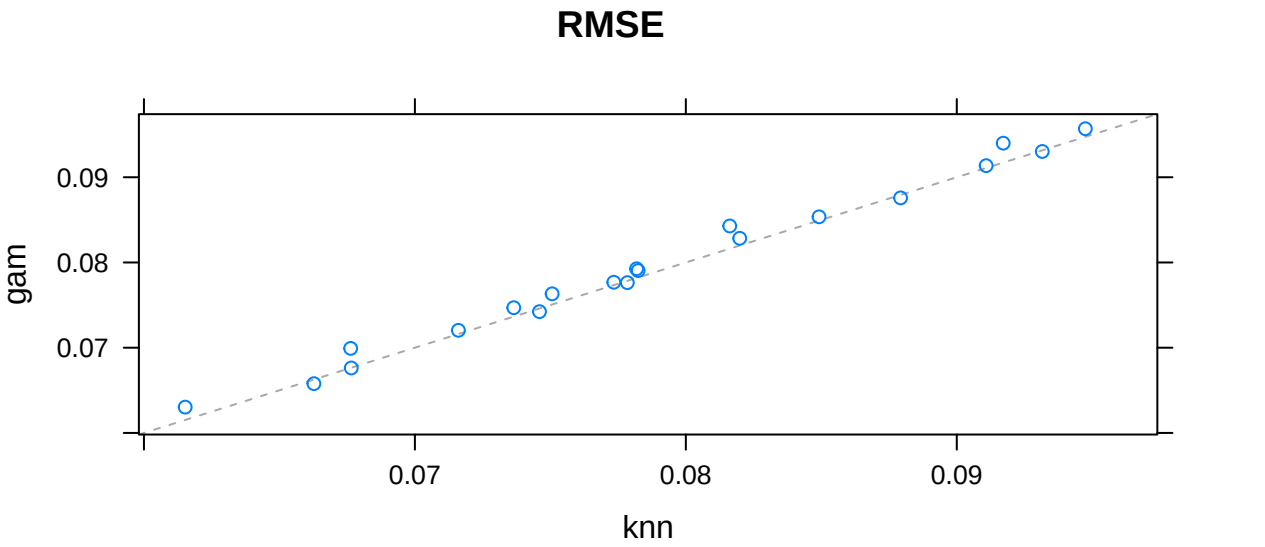
We would like to see if the folds where `knn` had problems are the same as the folds where `gamSpline` had issues. Fortunately `caret` allows us to investigate the RMSE for each of the 20 folds and compare how well `knn` did compared to `gamSpline`.

```
resamps <- resamples(list(gam=Spline, knn=knn))
summary(resamps)
```

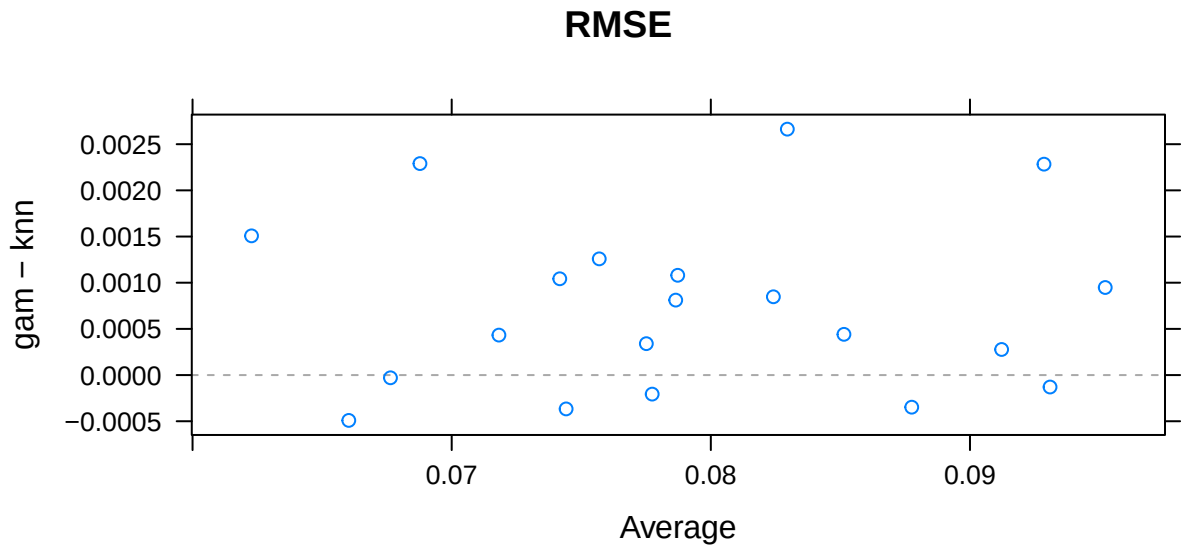
```
##
## Call:
## summary.resamples(object = resamps)
##
## Models: gam, knn
## Number of resamples: 20
##
## RMSE
##         Min. 1st Qu.  Median    Mean 3rd Qu.    Max. NA's
## gam 0.06303 0.07368 0.07837 0.07957 0.08591 0.09569    0
## knn 0.06152 0.07314 0.07801 0.07884 0.08567 0.09475    0
##
## Rsquared
##       Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## gam 0.8826  0.9127 0.9250 0.9222  0.9355 0.9453    0
```

```
## knn 0.8839  0.9138 0.9277 0.9242  0.9362 0.9493    0
```

```
xyplot(resamps)
```




```
xyplot(resamps, what='BlandAltman')
```



We could ask if the average difference could be equal to zero? With these twenty differences, we could just perform a t-test to make that comparison to see if the `gam` model typically has a greater difference in RMSE.

```
diff(resamps)  %>% summary()
```

```
##
## Call:
## summary.diff.resamples(object = .)
##
## p-value adjustment: bonferroni
## Upper diagonal: estimates of the difference
## Lower diagonal: p-value for H0: difference = 0
##
## RMSE
##     gam      knn
```

```
## gam          0.0007326
## knn 0.002217
##
## Rsquared
##     gam       knn
## gam           -0.001935
## knn 0.0003233
```

These two matricies give the point estimate of the difference in RMSE (gam - knn) as showing that there is weak evidence that knn predicts has a lower RMSE (difference = 0.0007326; p = 0.002217). Similarly we have weak evidence that the knn has a higher R-squared (difference = -0.001935; p = 0.0003233).

While these might be statistically significant, the practical difference of is pretty small, considering the y-values are spread across about 1-unit vertically, so the difference is about 1/10 of 1%. In this case, the two methods give practically identical inference.

5.2 Bootstrapping

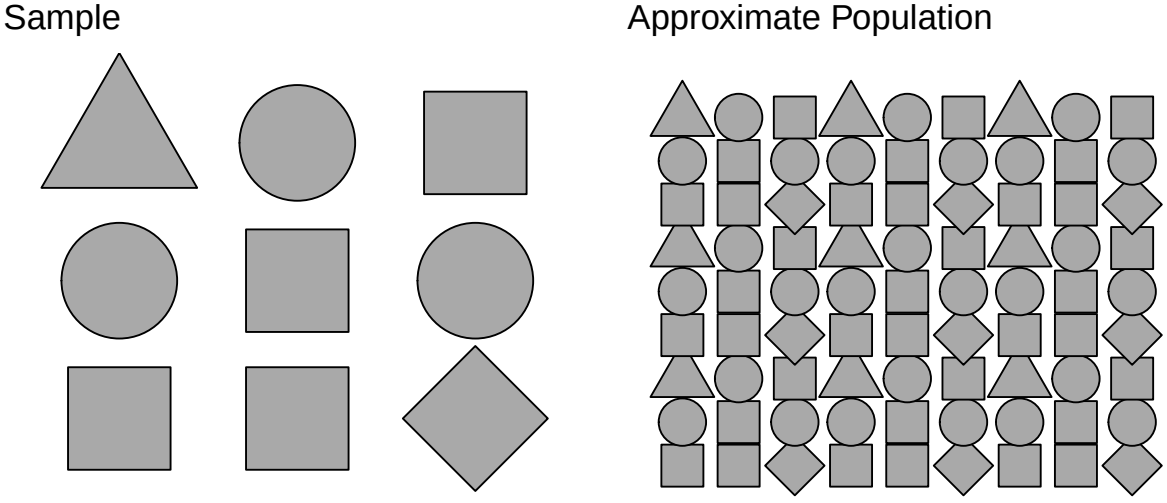
The basic goal of statistics is that we are interested in some population (which is described by some parameter μ, δ, τ, β, or generally, θ) and we take a random sample of size n from the population of interest and we truly believe that the sample is representative of the population of interest. Then we use some statistic of the data $\hat{\theta}$ as an estimate θ. However we know that this estimates, $\hat{\theta}$, vary from sample to sample. Previously we've used that the Central Limit Theorem gives

$$\hat{\theta} \dot{\sim} N\left(\theta, \sigma_{\hat{\theta}}\right)$$

to construct confidence intervals and perform hypothesis tests, but we don't necessarily like this approximation. If we could somehow take repeated samples (call these repeated samples $\mathbb{Y}_j$ for $j \in 1, 2, \ldots, M$) from the population we would understand the distribution of $\hat{\theta}$ by just examining the distribution of many observed values of $\hat{\theta}_j$ where $\hat{\theta}_j$ is the statistic calculated from the ith sample data $\mathbb{Y}_j$.

However, for practical reasons, we can't just take 1000s of samples of size n from the population. However, because we truly believe that $\mathbb{Y}$ is representative of the entire population, then our best guess of what the population is just many repeated copies of our data.

Suppose we were to sample from a population of shapes, and we observed 4/9 of the sample were squares, 3/9 were circles, and a triangle and a diamond. Then our best guess of what the population that we sampled from was a population with 4/9 squares, 3/9 circles, and 1/9 of triangles and diamonds.



Using this approximated population (which is just many many copies of our sample data), we can take many samples of size n. We denote these bootstrap samples as $\mathbb{Y}_j^*$, where the star denotes that the sample was taken from the approximate population, not the actual population. From each bootstrap sample $\mathbb{Y}_j^*$ a statistic of interest can be taken $\hat{\theta}_j^*$.

Because our approximate population is just an infinite number of copies of our sample data, then sampling from the approximate population is equivalent to sampling with replacement from our sample data. If I take n samples from n distinct objects with replacement, then the process can be thought of as mixing the n objects in a bowl and taking an object at random, noting which it is, replace it into the bowl, and then draw the next sample. Practically, this means some objects will be selected more than once and some will not be chosen at all. To sample our observed data with replacement, we'll use the `resample()` function in the `mosaic` package. We see that some rows will be selected multiple times, and some will not be selected at all.

5.2.1 Observational Studies vs Designed Experiments

The process of collecting data is a time consuming and laborious process but is critical to our understanding of the world. The fundamental goal is to collect a sample of dat that is representative of the population of interest and can provide insight into the scientific question at hand. There are two primary classes about how this data could be gathered, observational studies and designed experiments.

In an observational study, a population is identified and a random sample of individuals are selected to be in the sample. Then each subject in the sample has explanatory and response variables measured (fish are weighed and length recorded, people asked their age, gender, occupation etc). The critical part of this data collection method is that the random selection from the population is done in a fashion so that each individual in the population could potentially be in the sample and there is no systematic exclusion of certain parts of the population.

Simple Random Samples - Suppose that we could generate a list of every individual in the population and then we were to randomly select n of those to be our sample. Then each individual would have an equal chance to be in the sample and this selection scheme should result in sample data that is representative of the population of interest. Often though, it is difficult to generate a list of every individual, but other proxies might work. For example if we wanted to understand cougar behavior in the Grand Canyon, we might divide the park up into 100 regions and then random select 20 of those regions to sample and observe whatever cougar(s) are in that region.

Stratified Random Samples - In a stratified random sample, the population can be broken up into different strata and we perform a simple random sample within each strata. For example when sampling lake fish, we might think about the lake having deep and shallow/shore water strata and perhaps our sampling technique is different for those two strata (electro-fishing on shore and trawling in the deep sections). For human populations, we might stratify on age and geographic location (older retired people will answer the phone more readily than younger people). For each of the strata, we often have population level information about the different strata (proportion of the lake that is deep water versus shallow, or proportion of the population 20-29, 30-39, etc. and sample each strata accordingly (e.g. if shallow water is 40% of the fish habitat, then 40% of our sampling effort is spent in the shallows).

Regardless of sample type, the key idea behind an observational study is that we don't apply a treatment to the subject and then observe a response. While we might annoy animal or person, we don't do any long-term manipulations. Instead the individuals are randomly selected and then observed, and it is the random selection from the population that results in a sample that is representative of the population.

Designed Experiments - In an experimental setting, the subjects are taken from the population (usually not at random but rather by convenience) and then subjected to some treatments and we observe the individuals response to the treatment. There will usually be several levels of the treatment and there often is a control level. For example, we might want to understand how to maximize the growth of a type of fungus for a pharmaceutical application and we consider applying different nutrients to the substrate (nothing,

+phosphorus, +nitrogen, +both). Another example is researchers looking at the efficacy of smoking cessation methods and taking a set of willing subjects and having them try different methods (no help, nicotine patches, nicotine patches and a support group). There might be other covariates that we expect might affect the success rate (individuals age, length of time smoking, gender) and we might make sure that our study include people in each of these groups (we call these blocks in the experimental design terminology, but they are equivalent to the strata in the observational study terminology). Because even within blocks, we expect variability in the success rates due to natural variation, we randomize the treatment assignment to the individual and it is this randomization that addresses any unrecognized lurking variables that also affect the response.

A designed experiment is vastly superior to an observational experiment because the randomization of the treatment accounts for variables that the researcher might not even suspect to be important. A nice example of the difference between observational studies and experiments is a set of studies done relating breast cancer and hormone replacement therapy (HRT) drugs used by post-menopausal women. Initial observational studies that looked at the rates of breast cancer showed that women taking HRT had lower rates of breast cancer. When these results were first published, physicians happily recommended HRT to manage menopause symptoms and to decrease risk of breast cancer. Unfortunately subsequent observational studies showed a weaker effect and among some populations there was an increase in breast cancer. To answer the question clearly, a massive designed experiment was undertaken where women would be randomly assigned either a placebo or the actual HRT drugs. This study conclusively showed that HRT drugs increased the risk of breast cancer.

Why was there a disconnect between the original observational studies and the experiment? The explanation given is that there was a lurking variable that the observational studies did not control for… socio-economic class. There are many drivers of breast cancer and some of them are strongly correlated with socio-economic class such as where you live (in a polluted area or not). Furthermore because HRT was initially only to relieve symptoms of menopause, it wasn't "medically necessary" and insurance didn't cover it and so mainly wealthy women (with already lower risk for breast cancer) took the HRT drugs and the simple association between lower breast cancer risk and HRT was actually the effect of socio-economic status. By randomly assigning women to the placebo and HRT groups, high socio-economic women ended up in both groups. So even if there was some other lurking variable that the researchers didn't consider, the randomization would cause the unknown variable to be evenly distributed in the placebo and HRT groups.

Because the method of randomization is so different between observational studies and designed experiments, we should make certain that our method of creating bootstrap data sets respects that difference in randomization. So if there was some constraint on the data when it was originally taken, we want the bootstrap datasets to obey that same constraint. If our study protocol was to collect a sample of $n_1 = 10$ men and $n_2 = 10$ women, then we want our bootstrap samples to have 10 men and 10 women. If we designed an experiment with 25 subjects to test the efficacy of a drug and chose to administer doses of $5, 10, 20, 40$, and 80 mg with each five subjects for each dose level, then we want those same dose levels to show up in the bootstrap datasets.

There are two common approaches, *case resampling* and *residual resampling*. In case resampling, we consider the data (x_i, y_i) pairs as one unit and when creating a bootstrap sample, we resample those pairs, but if the ith data point is included in the bootstrap sample, then it is included as the (x_i, y_i) pair. In contrast, residual resampling is done by first fitting a model to the data, finding the residual values, resampling those residuals and then adding those bootstrap residuals to the predicted values $\hat{y}_i$.

```
Testing.Data <- data.frame(
  x = c(3,5,7,9),
  y = c(3,7,7,11))
Testing.Data
```

```
##   x  y
## 1 3  3
## 2 5  7
## 3 7  7
```

```
## 4 9 11
```

```
# Case resampling
Boot.Data <- mosaic::resample(Testing.Data)
Boot.Data
```

```
##     x  y orig.id
## 1   3  3       1
## 4   9 11       4
## 2   5  7       2
## 2.1 5  7       2
```

Notice that we've sampled $\{x = 5, y = 7\}$ twice and did not get the $\{7, 7\}$ data point.

Residual sampling is done by resampling the residuals and calling them $\hat{\epsilon}^*$ and then the new y-values will be $y_i^* = \hat{y}_i + \hat{\epsilon}_i^*$

```
# Residual resampling
model <- lm( y ~ x, data=Testing.Data)
Boot.Data <- Testing.Data %>%
  mutate( fit = fitted(model),
          resid = resid(model),
          resid.star = mosaic::resample(resid),
          y.star = fit + resid.star )
Boot.Data
```

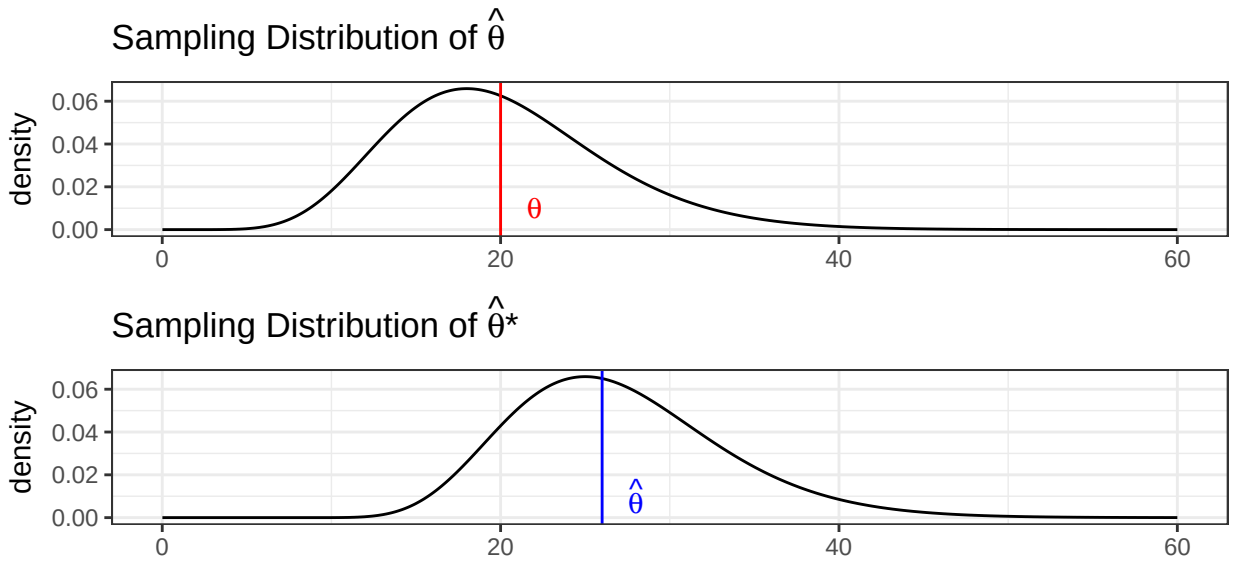
```
##   x  y  fit resid resid.star y.star
## 1 3  3  3.4  -0.4       -1.2    2.2
## 2 5  7  5.8   1.2       -1.2    4.6
## 3 7  7  8.2  -1.2       -0.4    7.8
## 4 9 11 10.6   0.4        1.2   11.8
```

Notice that the residuals resampling results in a data set where each of the x-values is retained, but a new y-value (possibly not seen in the original data) is created from the predicted value $\hat{y}$ and a randomly selected residual.

In general when we design an experiment, we choose which x-values we want to look at and so the bootstrap data should have those same x-values we chose. So for a designed experiment, we typically will create bootstrap data sets via residual resampling. For observational studies, we'll create the bootstrap data sets via case resampling. In both cases if there is a blocking or strata variable to consider, we will want to do the resampling within the block/strata.

5.2.2 Confidence Interval Types

We want to understand the relationship between the sample statistic $\hat{\theta}$ to the population parameter θ. We create an estimated population using many repeated copies of our data. By examining how the simulated $\hat{\theta}^*$ vary relative to $\hat{\theta}$, we will understand how possible $\hat{\theta}$ values vary relative to θ.



We will outline several methods for producing confidence intervals (in the order of most assumptions to fewest). For each of these methods, we will be interested in creating a $1-2\alpha$ confidence interval, so in all the formulas presented here, consider α to be the amount of probability in either tail and for a 95% CI, we will use $\alpha = 0.025$.

5.2.2.1 Normal intervals

This confidence interval assumes the sampling distribution of $\hat{\theta}$ is approximately normal (which is often true due to the central limit theorem). We can use the bootstrap replicate samples to get an estimate of the standard error of the statistic of interest by just calculating the sample standard deviation of the replicated statistics.

Let θ be the statistic of interest and $\hat{\theta}$ be the value of that statistic calculated from the observed data. Define $\hat{SE}^*$ as the sample standard deviation of the $\hat{\theta}^*$ values.

Our first guess as to a $(1-2\alpha)*100\%$ confidence interval is

$$\hat{\theta} \pm z_{1-\alpha}\hat{SE}^*$$

which we could write as

$$\left[\hat{\theta} - z_{1-\alpha}\hat{SE}^*, \quad \hat{\theta} + z_{1-\alpha}\hat{SE}^*\right]$$

5.2.2.2 Percentile intervals

The percentile interval doesn't assume normality but it does assume that the bootstrap distribution is symmetric and unbiased for the population value. This is the method we used to calculate confidences intervals in the first several chapters. It is perhaps the easiest to calculate and understand. This method only uses $\hat{\theta}^*$, and, for a $(1-2\alpha)*100\%$ confidence interval is:

$$\left[\hat{\theta}^*_{\alpha}, \quad \hat{\theta}^*_{1-\alpha}\right]$$

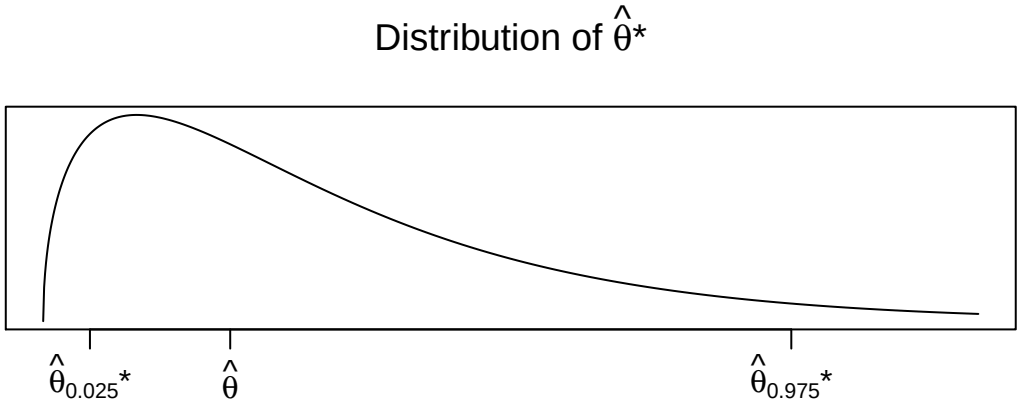
5.2.2.3 Basic intervals

Unlike the percentile bootstrap interval, the basic interval does not assume the bootstrap distribution is symmetric but does assume that $\hat{\theta}$ is an unbiased estimate for θ.

To address this, we will using the observed distribution of our replicates $\hat{\theta}^*$. Let $\hat{\theta}^*_\alpha$ and $\hat{\theta}^*_{1-\alpha}$ be the α and $1-\alpha$ quantiles of the replicates $\hat{\theta}^*$. Then another way to form a $(1-2\alpha)*100\%$ confidence interval would be

$$\left[\hat{\theta} - \left(\hat{\theta}^*_{1-\alpha} - \hat{\theta}\right), \quad \hat{\theta} - \left(\hat{\theta}^*_{\alpha} - \hat{\theta}\right)\right]$$

where the minus sign on the upper limit is because $\left(\hat{\theta}^*_\alpha - \hat{\theta}\right)$ is already negative. The idea behind this interval is that the sampling variability of $\hat{\theta}$ from θ is the same as the sampling variability of the replicates $\hat{\theta}^*$ from $\hat{\theta}$, and that the distribution of $\hat{\theta}$ is possibly skewed, so we can't add/subtract the same amounts. Suppose we observe the distribution of $\hat{\theta}^*$ as



Then any particular value of $\hat{\theta}^*$ could be much larger than $\hat{\theta}$. Therefore $\hat{\theta}$ could be much larger than θ. Therefore our confidence interval should be $\Big[\hat{\theta} - \text{big},\ \hat{\theta} + \text{small}\Big]$.

This formula can be simplified to

$$\left[\hat{\theta} - \left(\hat{\theta}^*_{1-\alpha/2} - \hat{\theta}\right)\ , \hat{\theta} + \left(\hat{\theta} - \hat{\theta}^*_{\alpha/2}\right)\right] = \left[2\hat{\theta} - \hat{\theta}^*_{1-\alpha/2}\ ,\ \ 2\hat{\theta} - \hat{\theta}^*_{\alpha/2}\right]$$

5.2.2.4 Bias-corrected and accelerated intervals (BCa)

Different schemes for creating confidence intervals can get quite complicated. There is a thriving research community investigating different ways of creating intervals and which are better in what instances. The BCa interval is a variation of the percentile method and is the most general of the bootstrap intervals and makes the fewest assumptions. The $(1-2\alpha)*100\%$ confidence interval is give by

$$\left[\hat{\theta}^*_{\alpha_1}, \hat{\theta}^*_{\alpha_2}\right]$$

where

$$\alpha_1 = \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z_\alpha}{1 - \hat{\alpha}(\hat{z}_0 + z_\alpha)}\right)$$

$$\alpha_2 = \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z_{1-\alpha}}{1 - \hat{\alpha}(\hat{z}_0 + z_{1-\alpha})}\right)$$

and $\Phi(\cdot)$ is the standard normal cdf, and z_α is the α quantile of the standard normal distribution. If $\hat{\alpha} = \hat{z}_0 = 0$, then these formulas reduce to

$$\alpha_1 = \Phi(z_\alpha) = \alpha \quad \text{and} \quad \alpha_2 = \Phi(z_{1-\alpha}) = 1-\alpha$$

The bias correction term, $\hat{z}_0$ is derived from the proportion of $\hat{\theta}^*$ values that are less than the original $\hat{\theta}$

$$\hat{z}_0 = \Phi^{-1}\left(\frac{\sum I(\hat{\theta}^*_i < \hat{\theta})}{B}\right)$$

where B is the number of bootstrap samples taken and $I()$ is the indicator function that takes on the value of 1 if the comparison is true and 0 otherwise.

The acceleration term, $\hat{\alpha}$, is calculated using the *jackknife estimate* of the statistic using the **original data**. Let $\hat{\theta}_{(i)}$ be the statistic of interest calculated using all but the ith observation from the original data and define the jackknife estimate $\hat{\theta}_{(\cdot)}$ as

$$\hat{\theta}_{(\cdot)} = \frac{1}{n}\sum_{i=1}^{N}\hat{\theta}_{(i)}$$

and finally we can calculate

$$\hat{a} = \frac{\sum_{i=1}^{n}\left(\hat{\theta}_{(\cdot)} - \hat{\theta}_{(i)}\right)^3}{6\left[\sum_{i=1}^{n}\left(\hat{\theta}_{(\cdot)} - \hat{\theta}_{(i)}\right)^2\right]^{3/2}}$$

It is not at all obvious why $\hat{a}$ is an appropriate term, but interested readers should consult chapter 14 of Efron and Tibshirani's *An Introduction to the Bootstrap*.

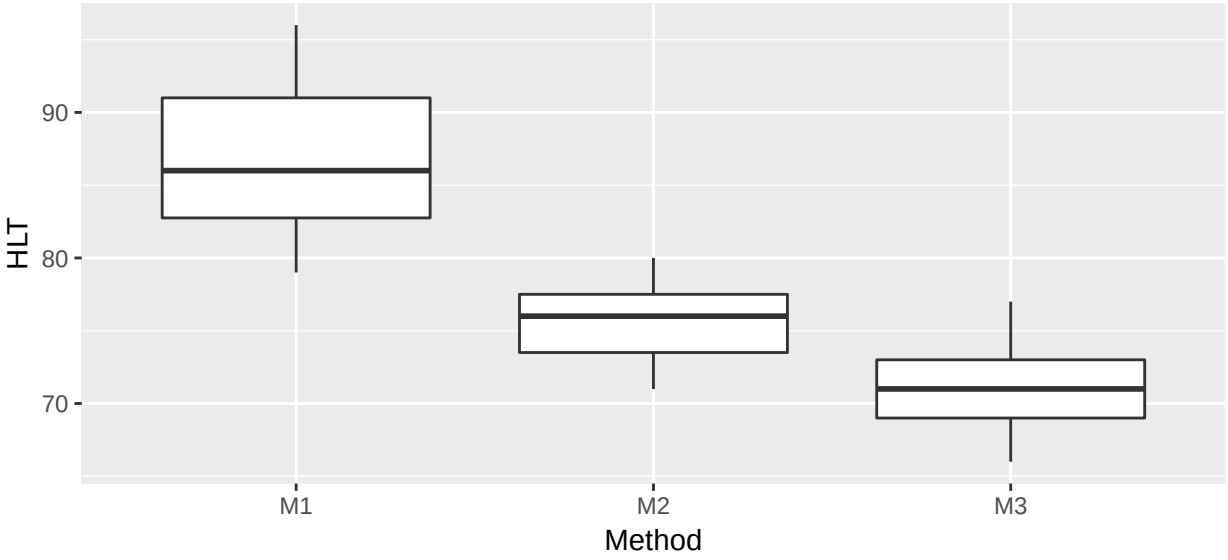
5.2.3 Using `car::Boot()` function

For every model we've examined we can create simulated data sets using either case or residual resampling and produce confidence intervals for any of the parameters of interest. We won't bother to do this by hand, but rather let R do the work for us. The package that contains most of the primary programs for bootstrapping is the package `boot`. The functions within this package are quite flexible but they are a little complex. While we will use this package directly later, for now we will use the package `car` which has a very convenient function `car::Boot()`.

We return to our ANOVA example of hostility scores after three different treatment methods. The first thing we will do (as we should do in all data analyses) is to graph our data.

```
# define the data
Hostility <- data.frame(
  HLT = c(96,79,91,85,83,91,82,87,
          77,76,74,73,78,71,80,
          66,73,69,66,77,73,71,70,74),
  Method = c( rep('M1',8), rep('M2',7), rep('M3',9) ) )
```

```
ggplot(Hostility, aes(x=Method, y=HLT)) +
  geom_boxplot()
```



We can fit the cell-means model and examine the summary statistics using the following code.

```
model <- lm( HLT ~ -1 + Method, data=Hostility )
summary(model)
```

```
##
## Call:
## lm(formula = HLT ~ -1 + Method, data = Hostility)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -7.750 -2.866  0.125  2.571  9.250
##
## Coefficients:
##          Estimate Std. Error t value Pr(>|t|)
## MethodM1   86.750      1.518   57.14   <2e-16 ***
## MethodM2   75.571      1.623   46.56   <2e-16 ***
## MethodM3   71.000      1.431   49.60   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.294 on 21 degrees of freedom
## Multiple R-squared:  0.9973, Adjusted R-squared:  0.997
## F-statistic:  2631 on 3 and 21 DF,  p-value: < 2.2e-16
```

Confidence intervals using the

$$\epsilon_{ij} \overset{iid}{\sim} N(0, \sigma)$$

assumption are given by

```
confint(model)
```

```
##             2.5 %   97.5 %
## MethodM1 83.59279 89.90721
## MethodM2 72.19623 78.94663
## MethodM3 68.02335 73.97665
```

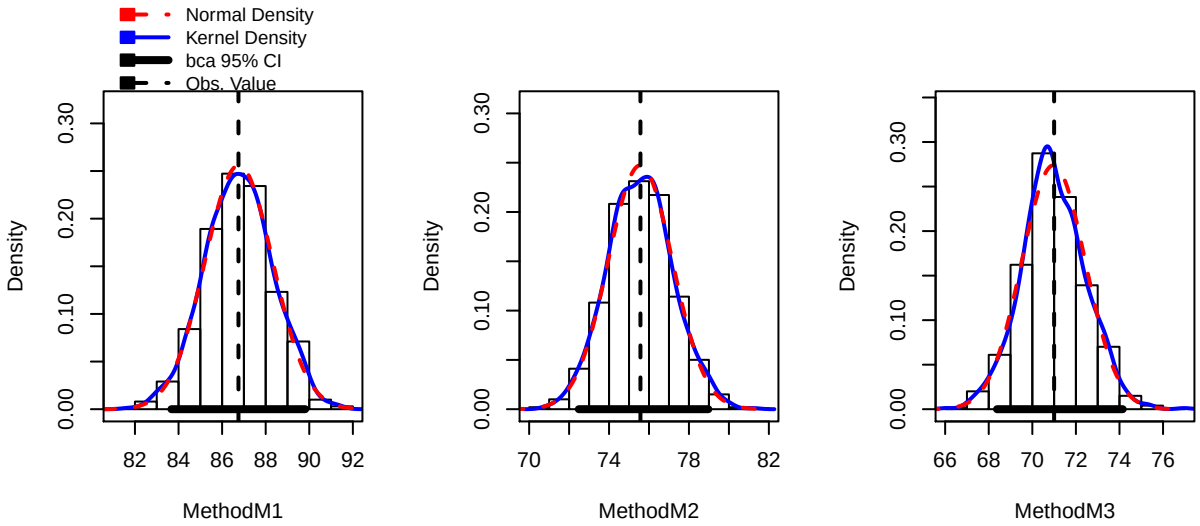
To utilize the bootstrap confidence intervals, we will use the function `car::Boot` from the package `car`. It defaults to using case resampling, but `method='residual'` will cause it to use residual resampling. We can

control the number of bootstrap replicates it using with the R parameter.

```
boot.model <- Boot(model, method='case',     R=999) # default case resampling
boot.model <- Boot(model, method='residual', R=999) # residual resampling
```

The `car::Boot()` function has done all work of doing the resampling and storing values of $\hat{\mu}_1, \hat{\mu}_2$, and $\hat{\mu}_3$ for each bootstrap replicate data set created using case resampling. To look at the bootstrap estimate of the sampling distribution of these statistics, we use the `hist()` function. The `hist()` function is actually overloaded and will act differently depending on the type of object. We will send it an object of class boot and the `hist()` function looks for a function name `hist.boot()` and when it finds it, just calls it with the function arguments we passed.

```
hist(boot.model, layout=c(1,3)) # 1 row, 3 columns of plots
```



While this plot is aesthetically displeasing (we could do so much better using ggplot2!) this shows the observed bootstrap histogram of $\hat{\mu}_i^*$, along with the normal distribution centered at $\hat{\mu}_i$ with spread equal to the $StdDev\,(\hat{\mu}_i^*)$. In this case, the sampling distribution looks very normal and the bootstrap confidence intervals should line up well with the asymptotic intervals. The function `confint()` will report the BCa intervals by default, but you can ask for "bca", "norm", "basic", "perc".

```
confint(boot.model)
```

```
## Bootstrap quantiles, type =  bca
##
##             2.5 %   97.5 %
## MethodM1 83.67302 89.81350
## MethodM2 72.48592 78.97034
## MethodM3 68.37861 74.14954
```

```
confint(boot.model, type='perc')
```

```
## Bootstrap quantiles, type =  percent
##
##             2.5 %   97.5 %
## MethodM1 83.66222 89.80938
## MethodM2 72.43179 78.82929
## MethodM3 68.04848 73.89276
```

```
confint(model)
```

```
##             2.5 %   97.5 %
```

```
## MethodM1 83.59279 89.90721
## MethodM2 72.19623 78.94663
## MethodM3 68.02335 73.97665
```

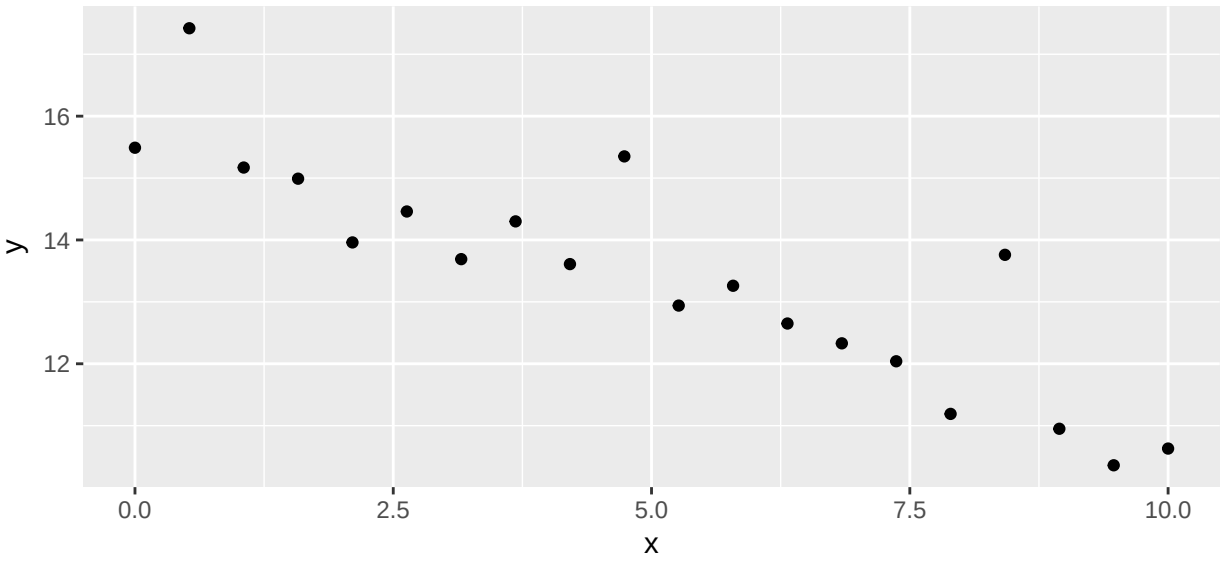
In this case we see that the confidence intervals match up very well with asymptotic intervals.

The `car::Boot()` function will work for a regression model as well. In the following example, the data was generated from

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

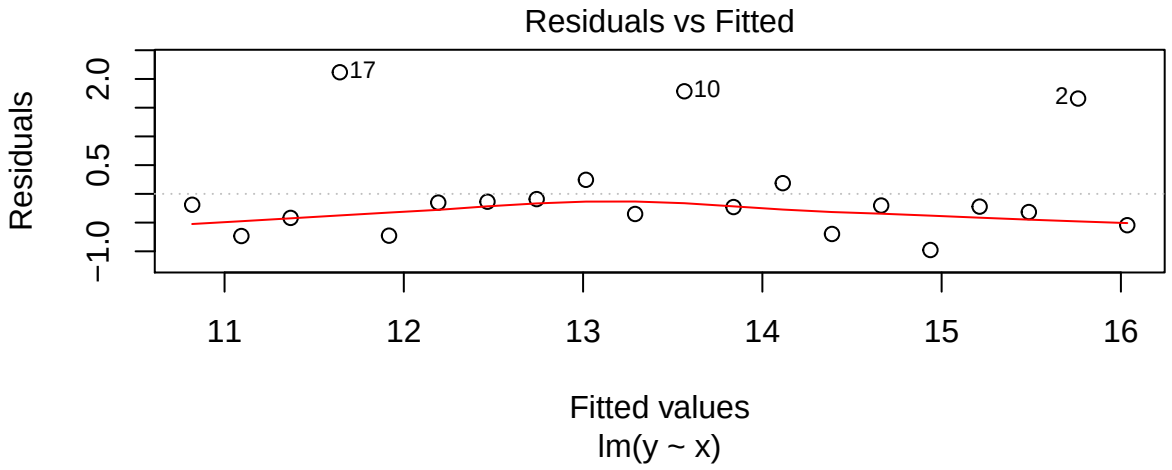
but the ϵ_i terms have a strong positive skew and are not normally distributed.

```
my.data <- data.frame(
  x = seq(0,10, length=20),
  y = c( 15.49, 17.42, 15.17, 14.99, 13.96,
         14.46, 13.69, 14.30, 13.61, 15.35,
         12.94, 13.26, 12.65, 12.33, 12.04,
         11.19, 13.76, 10.95, 10.36, 10.63))
ggplot(my.data, aes(x=x, y=y)) + geom_point()
```



Fitting a linear model, we see a problem that the residuals don't appear to be balanced. The large residuals are all positive. The Shapiro-Wilks test firmly rejects normality of the residuals.

```
model <- lm( y ~ x, data=my.data)
plot(model, which=1)
```

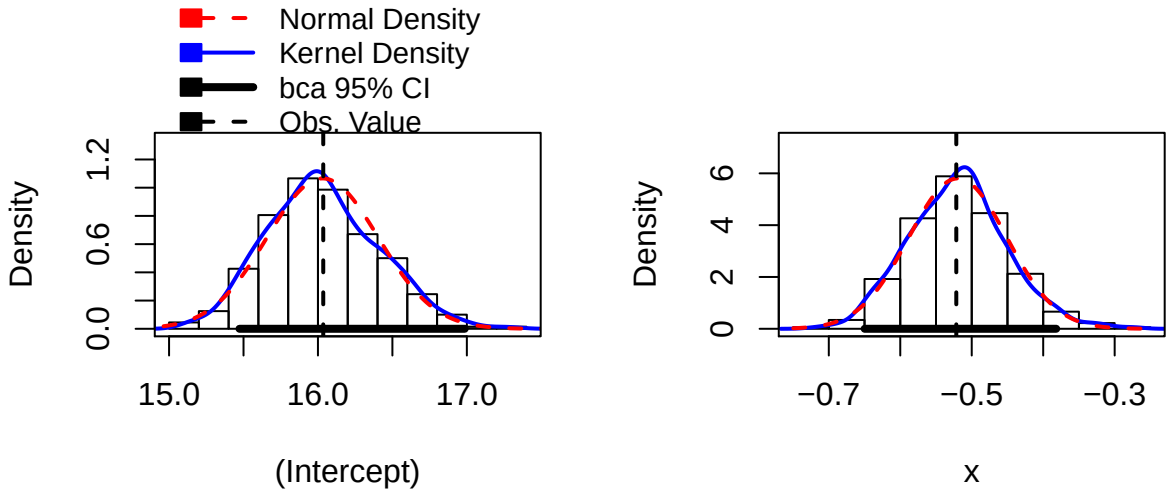



```
shapiro.test( resid(model) )
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(model)
## W = 0.77319, p-value = 0.0003534
```

As a result, we don't might not feel comfortable using the asymptotic distribution of $\hat{\beta}_0$ and $\hat{\beta}_1$ for the creation of our confidence intervals. The bootstrap procedure can give reasonable good intervals, however.

```
boot.model <- Boot( model )  # by default method='case'
hist( boot.model )
```




```
confint( boot.model )
```

```
## Bootstrap quantiles, type =  bca
##
##                 2.5 %     97.5 %
## (Intercept) 15.4729799 16.9857123
## x           -0.6498092 -0.3817759
```

Notice that both of the bootstrap distribution for both $\hat{\beta}_0^*$ and $\hat{\beta}_1^*$ are skewed, and the BCa intervals are likely to be the most appropriate intervals to use.

5.2.4 Using the boot package

The car::Boot() function is very handy, but it lacks flexibility; it assumes that you just want to create bootstrap confidence intervals for the model coefficients. The car::Boot() function is actually a nice simple user interface to the boot package which is more flexible, but requires the user to be more precise about what statistic should be stored and how the bootstrap samples should be created. We will next examine how to use this package.

5.2.4.1 Case resampling

Suppose that we have n observations in our sample data. Given some vector of numbers resampled from 1:n, we need to either resample those cases or those residuals and then using the new dataset calculate some statistic. The function boot() will require the user to write a function that does this.

```
model <- lm( y ~ x, data=my.data )
coef(model)
```

```
## (Intercept)           x
##  16.0355714  -0.5216143
```

```
# Do case resampling with the regression example
# sample.data is the original data frame
# indices - This is a vector of numbers from 1:n which tells
#           us which cases to use.  It might be 1,3,3,6,7,7,...
my.stat <- function(sample.data, indices){
  data.star <- sample.data[indices, ]
  model.star <- lm(y ~ x, data=data.star)
  output <- coef(model.star)
  return(output)
}

# original model coefficients
my.stat(my.data, 1:20)
```

```
## (Intercept)           x
##  16.0355714  -0.5216143
```

```
# one bootstrap replicate
my.stat(my.data, mosaic::resample(1:20))
```

```
## (Intercept)           x
##  16.3069314  -0.5786995
```

Notice that the function we write doesn't need to determine the random sample of the indices to use. Our function will be told what indices to use (possibly to calculate the statistic of interest $\hat{\theta}$, or perhaps a bootstrap replicate $\hat{\theta}^*$. For example, the BCa method needs to know the original sample estimates $\hat{\theta}$ to calculate how far the mean of the $\hat{\theta}^*$ values is from $\hat{\theta}$. To avoid the user having to see all of that, we just need to take the set of indices given and calculate the statistic of interest.

```
boot.model <- boot(my.data, my.stat, R=10000)
#boot.ci(boot.model, type='bca', index=1) # CI for Intercept
#boot.ci(boot.model, type='bca', index=2) # CI for the Slope
confint(boot.model)
```

```
## Bootstrap quantiles, type =  bca
##
##        2.5 %    97.5 %
```

```
## 1 15.4340761 17.013663
## 2 -0.6507547 -0.374097
```

5.2.4.2 Residual Resampling

We will now consider the ANOVA problem and in this case we will resample the residuals.

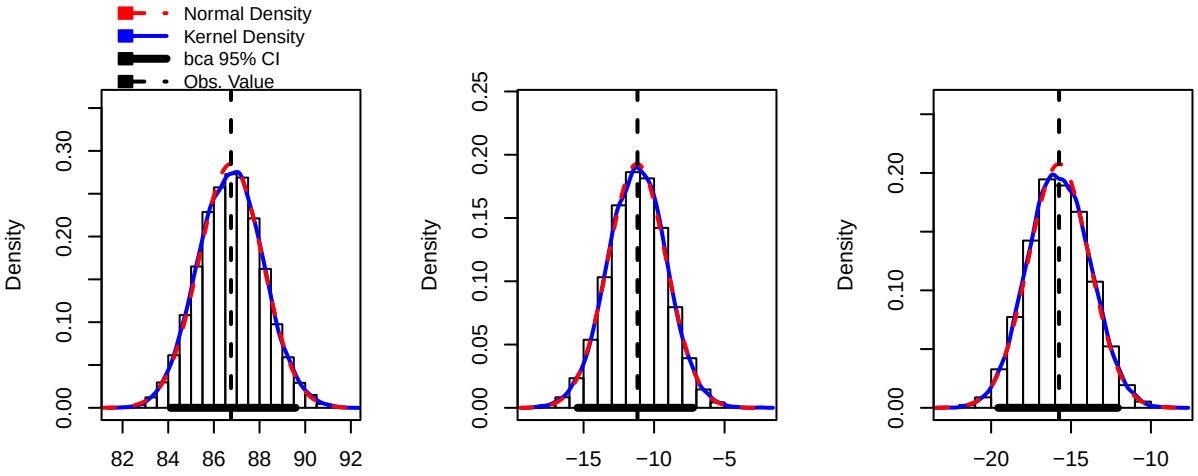
```
# Fit the ANOVA model to the Hostility Data
model <- lm( HLT ~ Method, data=Hostility )

# now include the predicted values and residuals to the data frame
Hostility <- Hostility %>% mutate(
  fit    = fitted(model),
  resid  = resid(model))

# Do residual resampling with the regression example
my.stat <- function(sample.data, indices){
  data.star  <- sample.data %>% mutate(HLT = fit + resid[indices])
  model.star <- lm(HLT ~ Method, data=data.star)
  output     <- coef(model.star)
  return(output)
}

boot.model <- boot(Hostility, my.stat, R=10000)
```

```
confint(boot.model)
```

```
## Bootstrap quantiles, type =  bca
##
##       2.5 %     97.5 %
## 1  84.10714  89.575893
## 2 -15.42106  -7.241673
## 3 -19.55282 -12.069872
```

Fortunately the `hist()` command can print the nice histogram from the output of the `boot()` command.

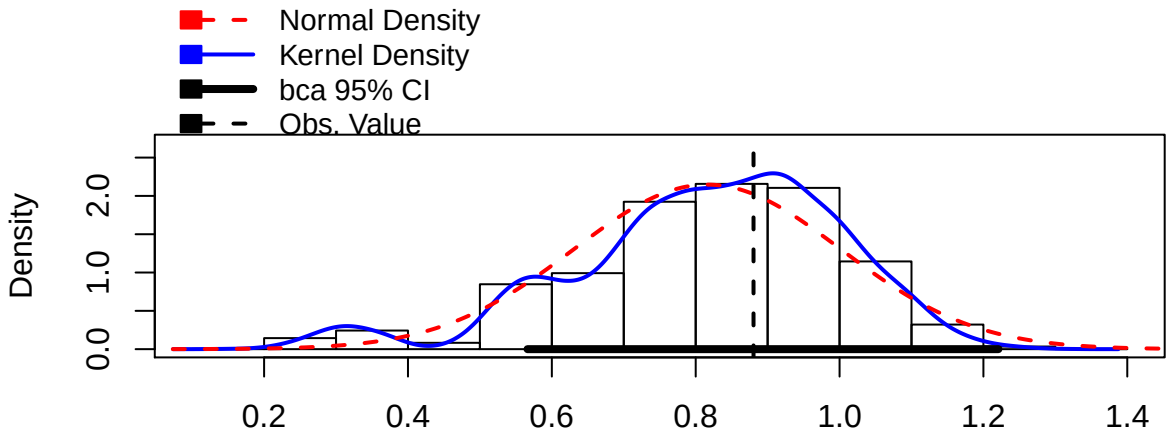
```
hist( boot.model, layout=c(1,3)) # 1 row, 3 columns)
```



Notice that we don't need to have the model coefficients $\hat{\mu}_i$ be our statistic of interest, we could just as easily produce a confidence interval for the residual standard error $\hat{\sigma}$.

```
# Do residual resampling with the regression example
model <- lm( y ~ x, data=my.data )
my.data <- my.data %>% mutate(
  fitted = fitted(model),
  resid  = resid(model))

# Define the statisitc I care about
my.stat <- function(sample.data, indices){
  data.star <- sample.data %>% mutate(y = fitted + resid[indices])
  model.star <- lm(y ~ x, data=data.star)
  output <- summary(model.star)$sigma
  return(output)
}

boot.model <- boot(my.data, my.stat, R=10000)
```

```
hist(boot.model, layout=c(1,3))
```




```
confint(boot.model)
```

```
## Bootstrap quantiles, type =  bca
##
##      2.5 %   97.5 %
## 1 0.566195 1.220835
```

5.2.4.3 Including Blocking/Stratifying Variables

When we introduced the ANOVA model we assumed that the groups had equal variance but we don't have to. If we consider the model with unequal variances among groups

$$Y_{ij} = \mu_i + \epsilon_{ij} \quad \text{where} \quad E(\epsilon_{ij}) = 0 \;\; Var(\epsilon_{ij}) = \sigma_i^2$$

then our usual analysis is inappropriate but we could easily bootstrap our confidence intervals for μ_i. If we do case resampling, this isn't an issue because each included observation is an (*group*, *response*) pair and our groups will have large or small variances similar to the observed data. However if we do residual resampling, then we must continue to have this. We do this by only resampling residuals within the same group. One way to think of this is if your model has a subscript on the variance term, then your bootstrap samples must respect that.

If you want to perform the bootstrap by hand using dplyr commands, it can be done by using the `group_by()` with whatever the blocking/Stratifying variable is prior to the `mosaic::resample()` command. You could

also use the optional group argument to the `mosaic::resample()` command.

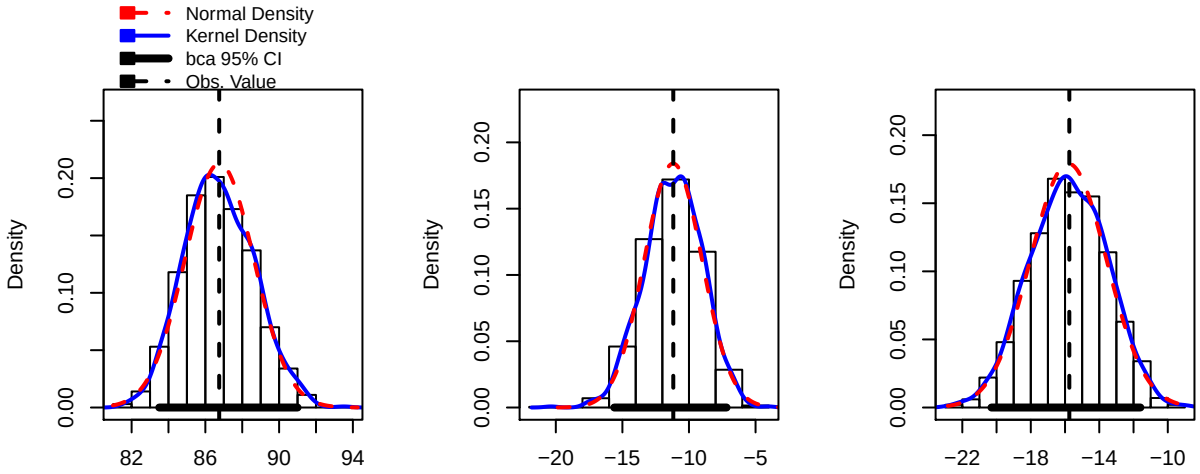
```
data <- data.frame(y  =c(9.8,9.9,10.1,10.2,   18,19,21,22),
                   grp=c('A','A','A','A',    'B','B','B','B'),
                   fit=c( 10,10,10,10,        20,20,20,20   ),
                   resid=c(-.2,-.1,.1,.2,     -2,-1,1,2   ))
data.star <- data %>%
  group_by(grp) %>%    # do the grouping using dplyr
  mutate(resid.star = mosaic::resample(resid),
         y.star     = fit + resid.star)
data.star
```

```
## # A tibble: 8 x 6
## # Groups:   grp [2]
##       y    grp   fit resid resid.star y.star
##   <dbl> <fctr> <dbl> <dbl>      <dbl>  <dbl>
## 1   9.8      A    10  -0.2        0.2   10.2
## 2   9.9      A    10  -0.1       -0.1    9.9
## 3  10.1      A    10   0.1       -0.2    9.8
## 4  10.2      A    10   0.2        0.2   10.2
## 5  18.0      B    20  -2.0       -2.0   18.0
## 6  19.0      B    20  -1.0        1.0   21.0
## 7  21.0      B    20   1.0       -1.0   19.0
## 8  22.0      B    20   2.0        1.0   21.0
```

Unfortunately the `car::Boot()` command doesn't take a strata option, but the the `boot::boot()` command.

```
# Fit the ANOVA model to the Hostility Data
model <- lm( HLT ~ Method, data=Hostility )

# now include the predicted values and residuals to the data frame
Hostility <- Hostility %>% mutate(
  fitted = fitted(model),
  resid  = resid(model))

# Do residual resampling
my.stat <- function(sample.data, indices){
  data.star <- sample.data %>% mutate(HLT = fitted + resid[indices])
  model.star <- lm(HLT ~ Method, data=data.star)
  output <- coef(model.star)
  return(output)
}

# strata is a vector of the categorical variable we block/stratify on
boot.model <- boot( Hostility, my.stat, R=1000, strata=Hostility$Method )
```

```
hist(boot.model, layout=c(1,3))
```




```
confint(boot.model)
```

```
## Bootstrap quantiles, type =  bca
##
##       2.5 %    97.5 %
## 1  83.50000  91.00000
## 2 -15.60967  -7.17667
## 3 -20.29828 -11.59939
```

5.3 Exercises

1. ISLR 5.3. We now review k-fold cross-validation.
   a) Explain how k-fold cross-validation is implemented.
   b) What are the advantages and disadvantages of k-fold cross validation relative to:
      i. The validation set approach?
      ii. LOOCV?
2. ISLR 5.2. We will now derive the probability that a given observation is part of a bootstrap sample. Suppose that we obtain a bootstrap sample from a set of n observations.
   a) What is the probability that the first bootstrap observation is not the jth observation from the original sample? Justify your answer.
   b) What is the probability that the second bootstrap observation is not the jth observation from the original sample?
   c) Argue that the probability that the jth observation is not in the bootstrap sample is (1-1/n)^{n}.
   d) When $n = 5$, what is the probability that the jth observation is in the bootstrap sample?
   e) When $n = 100$, what is the probability that the jth observation is in the bootstrap sample?
   f) When $n = 10,000$, what is the probability that the jth observation is in the bootstrap sample?
   g) Create a plot that displays, for each integer value of n from 1 to 100,000, the probability that the jth observation is in the bootstrap sample. Comment on what you observe.

   h) Investigate numerically the probability that a bootstrap sample of size n = 100 contains the jth observation. Here j = 4. Repeatedly create bootstrap samples, and each time we record whether or not the fourth observation is contained in the bootstrap sample. Comment on the results you obtain.
3. ISLR 5.7. In Sections 5.3.2 and 5.3.3, we saw that the cv.glm() function can be used in order to compute the LOOCV test error estimate. Alternatively, one could compute those quantities using just the `glm()` and `predict.glm()` functions, and a `for` loop. You will now take this approach in order to compute the LOOCV error for a simple logistic regression model on the `ISLR::Weekly` data set. Recall that in the context of classification problems, the LOOCV error is given in equation (5.4). The

context of this data set is the weekly percentage returns for the S&P 500 stock index between 1990 and 2010. In this problem we want to predict if the stock market is likely to go up or down depending on what it has done over the last two weeks.

a) Fit a logistic regression model that predicts `Direction` using `Lag1` and `Lag2`.
b) Fit a logistic regression model that predicts `Direction` using `Lag1` and `Lag2` using all but the first observation.
c) Use the model from (b) to predict the direction of the first observation. You can do this by predicting that the first observation will go up if P(Direction="Up" | Lag1, Lag2) > 0.5. Was this observation correctly classified?
d) Write a for loop from `i = 1` to `i = n`, where `n` is the number of observations in the data set, that performs each of the following steps:
   i. Fit a logistic regression model using all but the ith observation to predict Direction using Lag1 and Lag2.
   ii. Compute the posterior probability of the market moving up for the ith observation.
   iii. Use the posterior probability for the ith observation in order to predict whether or not the market moves up.
   iv. Determine whether or not an error was made in predicting the direction for the ith observation. If an error was made, then indicate this as a 1, and otherwise indicate it as a 0.
e) Take the average of the n numbers obtained in (d) in order to obtain the LOOCV estimate for the test error. Comment on the results.

4. We will now perform cross-validation on a simulated data set. The book has us performing this exercise using LOOCV, but we will use k-fold CV instead.

a) Generate a simulated data set as follows:

```
set.seed (1)
n <- 100
x <- rnorm(n)
y <- x - 2*x^2 + rnorm(n)
data <- data.frame(x=x, y=y)
```

In this data set, what are n and p? Write out the model used to generate the data in equation form.

b) Create a scatterplot of $\boldsymbol{x}$ against $\boldsymbol{y}$. Comment on what you find.
c) Compute the K-fold CV errors that result from fitting the following four models using least squares: *Hint: An arbitrary degree polynomial linear model can be fit using the following code:*

```
# fit a degree 2 polynomial
#  y = beta_0 + beta_1*x + beta_2*x^2
model <- lm( y ~ poly(x,2), data=data)
```

   i. $y = \beta_0 + \beta_1 x + \epsilon$
   ii. $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon$
   iii. $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon$
   iv. $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \epsilon$

d) Repeat step (c), and report your results. Are your results the same as what you got in (c)? Why?
e) Repeat step (c) using $k = 100$ folds. Notice this is LOOCV. If you repeat this analysis, will you get the same answer?
f) Which of the models had the smallest k-fold CV error? Which had the smallest LOOCV error? Is this what you expected? Explain your answer.
g) Comment on the statistical significance of the coefficient estimates that results from fitting each of the models in (c) using least squares. Do these results agree with the conclusions drawn based on the cross-validation results?

Chapter 6

Model Selection and Regularization

```
library(dplyr)    # data frame manipulations
library(ggplot2)  # plotting

library(caret)
library(glmnet)
```

6.1 Stepwise selection using AIC

Many researchers use forward or backward stepwise feature selection for both linear models or generalized linear models. There are a number of functions in R to facilitate this, notatbly `add1`, `drop1` and `step`.

We have a data set from the `faraway` package that has some information about each of the 50 US states. We'll use this to select a number of usefule covariates for predicting the states Life Expectancy.

```
library(faraway)
```

```
##
## Attaching package: 'faraway'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     wbca
```

```
## The following objects are masked from 'package:car':
##
##     logit, vif
```

```
## The following objects are masked from 'package:boot':
##
##     logit, melanoma
```

```
## The following object is masked from 'package:lattice':
##
##     melanoma
```

```
## The following object is masked from 'package:pracma':
##
##     logit
```

```
state.data <- state.x77 %>% data.frame() %>%
  mutate( State = rownames(.)) %>%
  mutate( HS.Grad.2 = HS.Grad^2,
          Income.2  = Income^2 )
# add a few squared terms to account for some curvature.
```

It is often necessary to compare models that are not nested. For example, I might want to compare

$$y = \beta_0 + \beta_1 x + \epsilon$$

vs

$$y = \beta_0 + \beta_2 w + \epsilon$$

This comparison comes about naturally when doing forward model selection and we are looking for the "best" covariate to add to the model first.

Akaike introduced his criterion (which he called "An Information Criterion") as

$$AIC = \underbrace{-2 \log L\left(\hat{\boldsymbol{\beta}}, \hat{\sigma} | \text{ data}\right)}_{\text{decreases if RSS decreases}} + \underbrace{2p}_{\text{increases as p increases}}$$

where $L\left(\hat{\boldsymbol{\beta}} | \text{ data}\right)$ is the likelihood function and p is the number of elements in the $\hat{\boldsymbol{\beta}}$ vector and we regard a lower AIC value as better. Notice the $2p$ term is essentially a penalty on adding addition covariates so to lower the AIC value, a new predictor must lower the negative log likelihood more than it increases the penalty.

To convince ourselves that the first summand decreases with decreasing RSS in the standard linear model, we examine the likelihood function

$$\begin{aligned} f\left(\boldsymbol{y} \,|\, \boldsymbol{\beta}, \sigma, \boldsymbol{X}\right) &= \frac{1}{\left(2\pi\sigma^2\right)^{n/2}} \exp\left[-\frac{1}{2\sigma^2}\left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right)^T\left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right)\right] \\ &= L\left(\boldsymbol{\beta}, \sigma \,|\, \boldsymbol{y}, \boldsymbol{X}\right) \end{aligned}$$

and we could re-write this as

$$\begin{aligned} \log L\left(\hat{\boldsymbol{\beta}}, \hat{\sigma} \,|\, \text{data}\right) &= -\log\left(\left(2\pi\hat{\sigma}^2\right)^{n/2}\right) - \frac{1}{2\hat{\sigma}^2}\left(\boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}\right)^T\left(\boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}\right) \\ &= -\frac{n}{2}\log\left(2\pi\hat{\sigma}^2\right) - \frac{1}{2\hat{\sigma}^2}\left(\boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}\right)^T\left(\boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}\right) \\ &= -\frac{1}{2}\left[n\log\left(2\pi\hat{\sigma}^2\right) + \frac{1}{\hat{\sigma}^2}\left(\boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}\right)^T\left(\boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}\right)\right] \\ &= -\frac{1}{2}\left[+n\log\left(2\pi\right) + n\log\hat{\sigma}^2 + \frac{1}{\hat{\sigma}^2}RSS\right] \end{aligned}$$

It isn't clear what we should do with the $n\log\left(2\pi\right)$ term in the $\log L()$ function. There are some compelling reasons to ignore it and just use the second, and there are reasons to use both terms. Unfortunately, statisticians have not settled on one convention or the other and different software packages might therefore report different values for AIC.

As a general rule of thumb, if the difference in AIC values is less than two then the models are not significantly different, differences between 2 and 4 AIC units are marginally significant and any difference greater than 4 AIC units is highly significant.

Notice that while this allows us to compare models that are not nested, it does require that the same data are used to fit both models. Because I could start out with my data frame including both x and x^2, (or

more generally x and $f(x)$ for some function $f()$) you can regard a transformation of a covariate as "the same data". However, a transformation of a y-variable is not and therefore we cannot use AIC to compare a models `log(y) ~ x` versus the model `y ~ x`.

Another criterion that might be used is *Bayes Information Criterion* (BIC) which is

$$BIC = -2 \log L\left(\hat{\beta}, \hat{\sigma} | \text{ data}\right) + p \log n$$

and this criterion punishes large models more than AIC does (because $\log n > 2$ for $n \geq 8$)

The AIC value of a linear model can be found using the AIC() on a lm() object.

```
m1 <- lm(Life.Exp ~ Income + Income.2 + Murder + Frost, data=state.data)
m2 <- lm(Life.Exp ~ Illiteracy + Murder + Frost, data=state.data)

AIC(m1)
```

```
## [1] 121.4293
```

```
AIC(m2)
```

```
## [1] 124.2947
```

Because the AIC value for the first model is lower, we would prefer the first model that includes both `Income` and `Income.2` compared to model 2, which was `Life.Exp ~ Illiteracy+Murder+Frost`.

6.1.1 Adjusted `R-sq`

One of the problems with R^2 is that it makes no adjustment for how many parameters in the model. Recall that R^2 was defined as

$$R^2 = \frac{RSS_S - RSS_C}{RSS_S} = 1 - \frac{RSS_C}{RSS_S}$$

where the simple model was the intercept only model. We can create an R^2_{adj} statistic that attempts to add a penalty for having too many parameters by defining

$$R^2_{adj} = 1 - \frac{RSS_C / (n-p)}{RSS_S / (n-1)}$$

With this adjusted definition, adding a variable to the model that has no predictive power will decrease R^2_{adj}.

6.1.2 Example

Returning to the life expectancy data, we could start with a simple model add covariates to the model that have the lowest AIC values. R makes this easy with the function `add1()` which will take a linear model (which includes the data frame that originally defined it) and will sequentially add all of the possible terms that are not currently in the model and report the AIC values for each model.

```
# Define the biggest model I wish to consider
biggest <- Life.Exp ~ Population + Income + Illiteracy + Murder +
                      HS.Grad + Frost + Area + HS.Grad.2 + Income.2

# Define the model I wish to start with
m <- lm(Life.Exp ~ 1, data=state.data)

add1(m, scope=biggest)  # what is the best addition to make?
```

```
## Single term additions
##
## Model:
## Life.Exp ~ 1
##            Df Sum of Sq    RSS     AIC
## <none>                  88.299  30.435
## Population  1     0.409 87.890  32.203
## Income      1    10.223 78.076  26.283
## Illiteracy  1    30.578 57.721  11.179
## Murder      1    53.838 34.461 -14.609
## HS.Grad     1    29.931 58.368  11.737
## Frost       1     6.064 82.235  28.878
## Area        1     1.017 87.282  31.856
## HS.Grad.2   1    27.414 60.885  13.848
## Income.2    1     7.464 80.835  28.020
```

Clearly the additiona of `Murder` to the model results in the lowest AIC value, so we will add `Murder` to the model. Notice the `<none>` row corresponds to the model m which we started with and it has a `RSS=88.299`. For each model considered, R will calculate the `RSS_{C}` for the new model and will calculate the difference between the starting model and the more complicated model and display this in the Sum of Squares column.

```
m <- update(m, . ~ . + Murder)  # add murder to the model
add1(m, scope=biggest)          # what should I add next?
```

```
## Single term additions
##
## Model:
## Life.Exp ~ Murder
##            Df Sum of Sq    RSS     AIC
## <none>                  34.461 -14.609
## Population  1    4.0161 30.445 -18.805
## Income      1    2.4047 32.057 -16.226
## Illiteracy  1    0.2732 34.188 -13.007
## HS.Grad     1    4.6910 29.770 -19.925
## Frost       1    3.1346 31.327 -17.378
## Area        1    0.4697 33.992 -13.295
## HS.Grad.2   1    4.4396 30.022 -19.505
## Income.2    1    1.8972 32.564 -15.441
```

There is a companion function to `add1()` that finds the best term to drop. It is conveniently named `drop1()` but here the `scope` parameter defines the smallest model to be considered.

It would be nice if all of this work was automated. Again, R makes our life easy and the function `step()` does exactly this. The set of models searched is determined by the scope argument which can be a *list* of two formulas with components upper and lower or it can be a single formula, or it can be blank. The right-hand-side of its lower component defines the smallest model to be considered and the right-hand-side of the upper component defines the largest model to be considered. If `scope` is a single formula, it specifies the upper component, and the lower model taken to be the intercept-only model. If scope is missing, the initial model is used as the upper model.

```
smallest <- Life.Exp ~ 1
biggest <- Life.Exp ~ Population + Income + Illiteracy +
                      Murder + HS.Grad + Frost + Area + HS.Grad.2 + Income.2
m <- lm(Life.Exp ~ Income, data=state.data)
step(m, scope=list(lower=smallest, upper=biggest))
```

```
## Start:  AIC=26.28
```

```
## Life.Exp ~ Income
##
##              Df Sum of Sq    RSS     AIC
## + Murder      1    46.020 32.057 -16.226
## + Illiteracy  1    21.109 56.968  12.523
## + HS.Grad     1    19.770 58.306  13.684
## + Income.2    1    19.062 59.015  14.288
## + HS.Grad.2   1    17.193 60.884  15.847
## + Area        1     5.426 72.650  24.682
## + Frost       1     3.188 74.889  26.199
## <none>                    78.076  26.283
## + Population  1     1.781 76.295  27.130
## - Income      1    10.223 88.299  30.435
##
## Step:  AIC=-16.23
## Life.Exp ~ Income + Murder
##
##              Df Sum of Sq    RSS     AIC
## + Frost       1     3.918 28.138 -20.745
## + Income.2    1     3.036 29.021 -19.200
## + Population  1     2.552 29.504 -18.374
## + HS.Grad     1     2.388 29.668 -18.097
## + HS.Grad.2   1     2.199 29.857 -17.780
## <none>                    32.057 -16.226
## - Income      1     2.405 34.461 -14.609
## + Illiteracy  1     0.011 32.046 -14.242
## + Area        1     0.000 32.057 -14.226
## - Murder      1    46.020 78.076  26.283
##
## Step:  AIC=-20.74
## Life.Exp ~ Income + Murder + Frost
##
##              Df Sum of Sq    RSS     AIC
## + HS.Grad     1     2.949 25.189 -24.280
## + HS.Grad.2   1     2.764 25.375 -23.914
## + Income.2    1     2.017 26.121 -22.465
## + Population  1     1.341 26.797 -21.187
## <none>                    28.138 -20.745
## + Illiteracy  1     0.950 27.189 -20.461
## + Area        1     0.147 27.991 -19.007
## - Income      1     3.188 31.327 -17.378
## - Frost       1     3.918 32.057 -16.226
## - Murder      1    46.750 74.889  26.199
##
## Step:  AIC=-24.28
## Life.Exp ~ Income + Murder + Frost + HS.Grad
##
##              Df Sum of Sq    RSS     AIC
## + Population  1     1.887 23.302 -26.174
## + Income.2    1     1.864 23.326 -26.124
## - Income      1     0.182 25.372 -25.920
## <none>                    25.189 -24.280
## + HS.Grad.2   1     0.218 24.972 -22.714
## + Illiteracy  1     0.131 25.058 -22.541
```

```
## + Area         1     0.058 25.131 -22.395
## - HS.Grad      1     2.949 28.138 -20.745
## - Frost        1     4.479 29.668 -18.097
## - Murder       1    32.877 58.067  15.478
##
## Step:  AIC=-26.17
## Life.Exp ~ Income + Murder + Frost + HS.Grad + Population
##
##              Df Sum of Sq    RSS     AIC
## - Income      1     0.006 23.308 -28.161
## <none>                    23.302 -26.174
## + Income.2    1     0.790 22.512 -25.899
## - Population  1     1.887 25.189 -24.280
## + HS.Grad.2   1     0.006 23.296 -24.187
## + Illiteracy  1     0.004 23.298 -24.182
## + Area        1     0.000 23.302 -24.174
## - Frost       1     3.037 26.339 -22.048
## - HS.Grad     1     3.495 26.797 -21.187
## - Murder      1    34.739 58.041  17.456
##
## Step:  AIC=-28.16
## Life.Exp ~ Murder + Frost + HS.Grad + Population
##
##              Df Sum of Sq    RSS     AIC
## <none>                    23.308 -28.161
## + Income.2    1     0.031 23.277 -26.229
## + HS.Grad.2   1     0.007 23.301 -26.177
## + Income      1     0.006 23.302 -26.174
## + Illiteracy  1     0.004 23.304 -26.170
## + Area        1     0.001 23.307 -26.163
## - Population  1     2.064 25.372 -25.920
## - Frost       1     3.122 26.430 -23.877
## - HS.Grad     1     5.112 28.420 -20.246
## - Murder      1    34.816 58.124  15.528

##
## Call:
## lm(formula = Life.Exp ~ Murder + Frost + HS.Grad + Population,
##     data = state.data)
##
## Coefficients:
## (Intercept)       Murder        Frost      HS.Grad   Population
##   7.103e+01   -3.001e-01   -5.943e-03    4.658e-02    5.014e-05
```

Notice that our model selected by `step()` is not the same model we obtained when we started with the biggest model and removed things based on p-values.

The log-likelihood is only defined up to an additive constant, and there are different conventional constants used. This is more annoying than anything because all we care about for model selection is the difference between AIC values of two models and the additive constant cancels. The only time it matters is when you have two different ways of extracting the AIC values. Recall the model we fit using the top-down approach was

```
# m1 was
m1 <- lm(Life.Exp ~ Income + Murder + Frost + Income.2, data = state.data)
AIC(m1)
```

```
## [1] 121.4293
```

and the model selected by the stepwise algorithm was

```
m3 <- lm(Life.Exp ~ Murder + Frost + HS.Grad + Population, data = state.data)
AIC(m3)
```

```
## [1] 115.7326
```

Because `step()` and `AIC()` are following different conventions the absolute value of the AICs are different, but the difference between the two is constant no matter which function we use.

First we calculate the difference using the AIC() function:

```
AIC(m1) - AIC(m3)
```

```
## [1] 5.696681
```

and next we use `add1()` on both models to see what the AIC values for each.

```
add1(m1, scope=biggest)
```

```
## Single term additions
##
## Model:
## Life.Exp ~ Income + Murder + Frost + Income.2
##            Df Sum of Sq    RSS     AIC
## <none>                  26.121 -22.465
## Population  1   0.42412 25.697 -21.283
## Illiteracy  1   0.10097 26.020 -20.658
## HS.Grad     1   2.79527 23.326 -26.124
## Area        1   1.69309 24.428 -23.815
## HS.Grad.2   1   2.79698 23.324 -26.127
```

```
add1(m3, scope=biggest)
```

```
## Single term additions
##
## Model:
## Life.Exp ~ Murder + Frost + HS.Grad + Population
##            Df Sum of Sq    RSS     AIC
## <none>                  23.308 -28.161
## Income      1 0.0060582 23.302 -26.174
## Illiteracy  1 0.0039221 23.304 -26.170
## Area        1 0.0007900 23.307 -26.163
## HS.Grad.2   1 0.0073439 23.301 -26.177
## Income.2    1 0.0314248 23.277 -26.229
```

Using these results, we can calculate the difference in AIC values to be the same as we calculated before

$$\begin{aligned} -22.465 - -28.161 &= -22.465 + 28.161 \\ &= 5.696 \end{aligned}$$

```
smallest <- Life.Exp ~ 1
biggest  <- Life.Exp ~ Population + Income + Illiteracy +
                       Murder + HS.Grad + Frost + Area
m <- lm(Life.Exp ~ Income, data=state.data)
step(m, scope=list(lower=smallest, upper=biggest))
```

```
## Start:  AIC=26.28
```

```
## Life.Exp ~ Income
##
##              Df Sum of Sq    RSS     AIC
## + Murder      1    46.020 32.057 -16.226
## + Illiteracy  1    21.109 56.968  12.523
## + HS.Grad     1    19.770 58.306  13.684
## + Area        1     5.426 72.650  24.682
## + Frost       1     3.188 74.889  26.199
## <none>                    78.076  26.283
## + Population  1     1.781 76.295  27.130
## - Income      1    10.223 88.299  30.435
##
## Step:  AIC=-16.23
## Life.Exp ~ Income + Murder
##
##              Df Sum of Sq    RSS     AIC
## + Frost       1     3.918 28.138 -20.745
## + Population  1     2.552 29.504 -18.374
## + HS.Grad     1     2.388 29.668 -18.097
## <none>                    32.057 -16.226
## - Income      1     2.405 34.461 -14.609
## + Illiteracy  1     0.011 32.046 -14.242
## + Area        1     0.000 32.057 -14.226
## - Murder      1    46.020 78.076  26.283
##
## Step:  AIC=-20.74
## Life.Exp ~ Income + Murder + Frost
##
##              Df Sum of Sq    RSS     AIC
## + HS.Grad     1     2.949 25.189 -24.280
## + Population  1     1.341 26.797 -21.187
## <none>                    28.138 -20.745
## + Illiteracy  1     0.950 27.189 -20.461
## + Area        1     0.147 27.991 -19.007
## - Income      1     3.188 31.327 -17.378
## - Frost       1     3.918 32.057 -16.226
## - Murder      1    46.750 74.889  26.199
##
## Step:  AIC=-24.28
## Life.Exp ~ Income + Murder + Frost + HS.Grad
##
##              Df Sum of Sq    RSS     AIC
## + Population  1     1.887 23.302 -26.174
## - Income      1     0.182 25.372 -25.920
## <none>                    25.189 -24.280
## + Illiteracy  1     0.131 25.058 -22.541
## + Area        1     0.058 25.131 -22.395
## - HS.Grad     1     2.949 28.138 -20.745
## - Frost       1     4.479 29.668 -18.097
## - Murder      1    32.877 58.067  15.478
##
## Step:  AIC=-26.17
## Life.Exp ~ Income + Murder + Frost + HS.Grad + Population
##
```

```
##              Df Sum of Sq    RSS     AIC
## - Income      1     0.006 23.308 -28.161
## <none>                    23.302 -26.174
## - Population  1     1.887 25.189 -24.280
## + Illiteracy  1     0.004 23.298 -24.182
## + Area        1     0.000 23.302 -24.174
## - Frost       1     3.037 26.339 -22.048
## - HS.Grad     1     3.495 26.797 -21.187
## - Murder      1    34.739 58.041  17.456
##
## Step:  AIC=-28.16
## Life.Exp ~ Murder + Frost + HS.Grad + Population
##
##              Df Sum of Sq    RSS     AIC
## <none>                    23.308 -28.161
## + Income      1     0.006 23.302 -26.174
## + Illiteracy  1     0.004 23.304 -26.170
## + Area        1     0.001 23.307 -26.163
## - Population  1     2.064 25.372 -25.920
## - Frost       1     3.122 26.430 -23.877
## - HS.Grad     1     5.112 28.420 -20.246
## - Murder      1    34.816 58.124  15.528
```

```
##
## Call:
## lm(formula = Life.Exp ~ Murder + Frost + HS.Grad + Population,
##     data = state.data)
##
## Coefficients:
## (Intercept)       Murder        Frost      HS.Grad   Population
##   7.103e+01   -3.001e-01   -5.943e-03    4.658e-02    5.014e-05
```

This same approach works for `glm` objects as well. Unfortunately there isn't a way to make this work via the `caret` package, and so we can't do quite the same thing in general.

6.2 Model Regularization via LASSO and Ridge Regression

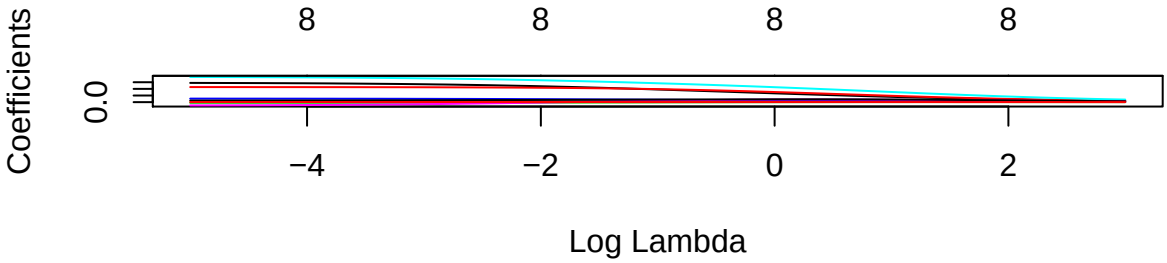
For linear models we might consider adding a penalty to the function we seek to minimize. By minimizing adding a penalty in the form of either $\sum |\beta_j|$ or $\sum \beta_j^2$ we get either ridge regression or LASSO.

For this example, we'll consider data from a study about prostate cancer and we are interested in predicting a prostate specific antigen that is highly elevated in cancerous tumors.

6.2.1 Ridge Regression

```
ctrl <- trainControl( method='repeatedcv', number=5, repeats=4,
                      preProcOptions = c('center','scale'))
grid <- data.frame(
  alpha  = 0,  # 0 => Ridge Regression
  lambda = exp(seq(-5, 3, length=100)) )

model <- train( lpsa ~ ., data=prostate, method='glmnet',
```

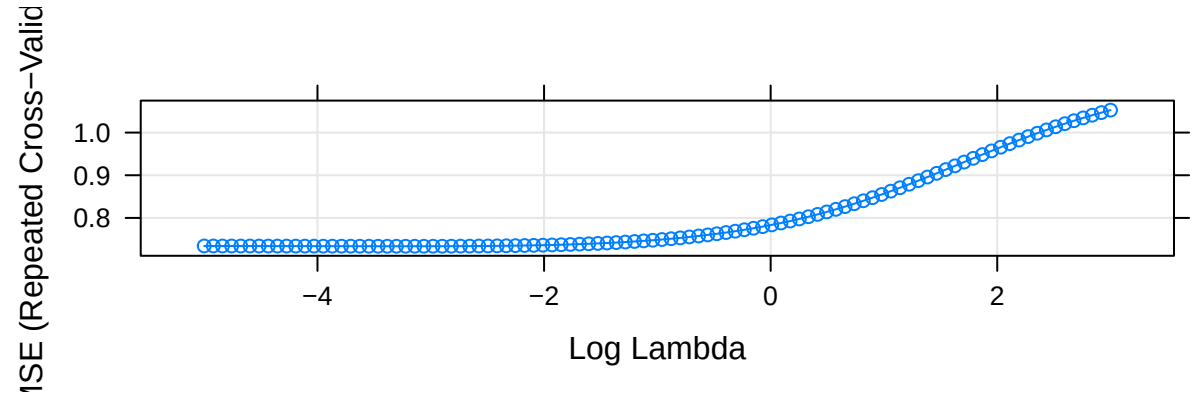
```
                trControl=ctrl, tuneGrid=grid,
                lambda= grid$lambda )    # Not sure why lambda isn't being passed in...

plot.glmnet(model$finalModel, xvar='lambda')
```



Each line corresponds to the β_j coefficient for each λ value. The number at the top is the number of non-zero coefficients for that particular λ value.

Next we need to figure out the best value of λ that we considered.

```
plot(model, xTrans = log, xlab='Log Lambda')
```



So based on this graph, we want to choose λ to be as large a possible without increasing RMSE too much. So $\log(\lambda) \approx -2$ seems about right. And therefore $\lambda \approx e^{-2.35} = 0.095$

```
model$bestTune
```

```
##    alpha     lambda
## 23     0 0.03986637
```

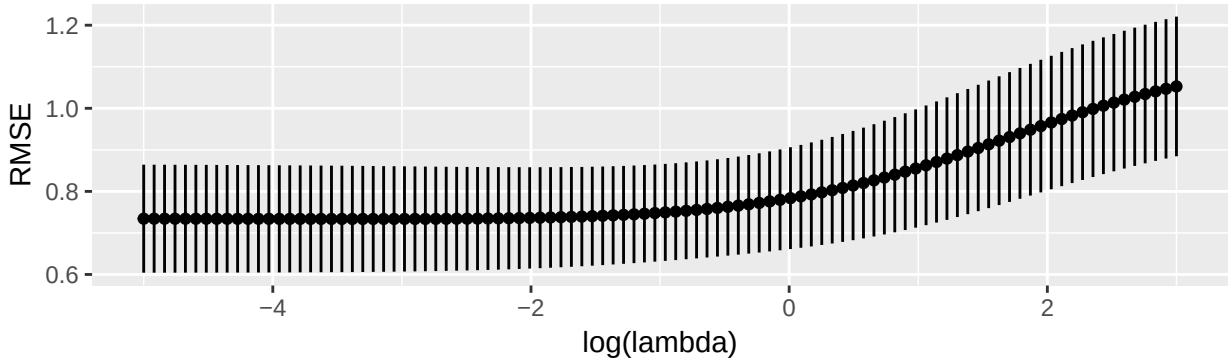
The problem is that we will observe different "bestTune" values if we repeat the analysis. It would be nice if this graph showed the standard errors of the response.

```
str( model$results )
```

```
## 'data.frame':    100 obs. of  8 variables:
##  $ alpha     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ lambda    : num  0.00674 0.00731 0.00792 0.00859 0.00931 ...
##  $ RMSE      : num  0.734 0.734 0.734 0.734 0.734 ...
##  $ Rsquared  : num  0.612 0.612 0.612 0.612 0.612 ...
##  $ MAE       : num  0.574 0.574 0.574 0.574 0.574 ...
##  $ RMSESD    : num  0.13 0.13 0.13 0.13 0.13 ...
##  $ RsquaredSD: num  0.13 0.13 0.13 0.13 0.13 ...
##  $ MAESD     : num  0.122 0.122 0.122 0.122 0.122 ...
```

```
ggplot(model$results, aes( x=log(lambda) )) +
  geom_point( aes(y=RMSE) ) +
```

```
  geom_line( aes(y=RMSE) ) +
  geom_linerange(aes( ymin= RMSE - RMSESD, ymax= RMSE+RMSESD))
```



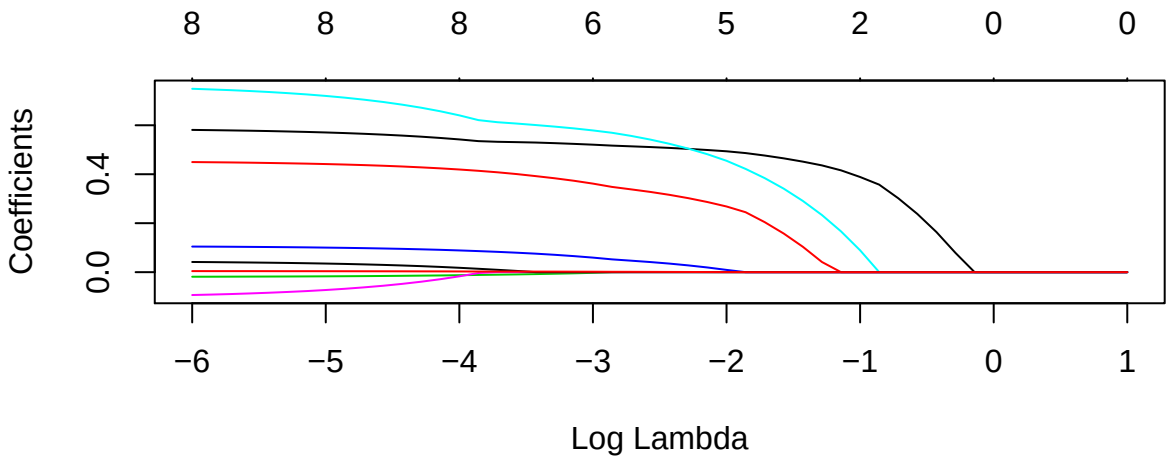
Given this, I feel ok chosing anthing from about $\log(\lambda) \in [-2, -1]$

6.2.2 Lasso

```
ctrl <- trainControl( method='repeatedcv', number=5, repeats=4,
                      preProcOptions = c('center','scale'))
grid <- data.frame(
  alpha  = 1,  # 1 => Lasso Regression
  lambda = exp(seq(-6, 1, length=50)))

model <- train( lpsa ~ ., data=prostate, method='glmnet',
                trControl=ctrl, tuneGrid=grid,
                lambda = grid$lambda )
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.
```

```
plot.glmnet(model$finalModel, xvar='lambda')
```

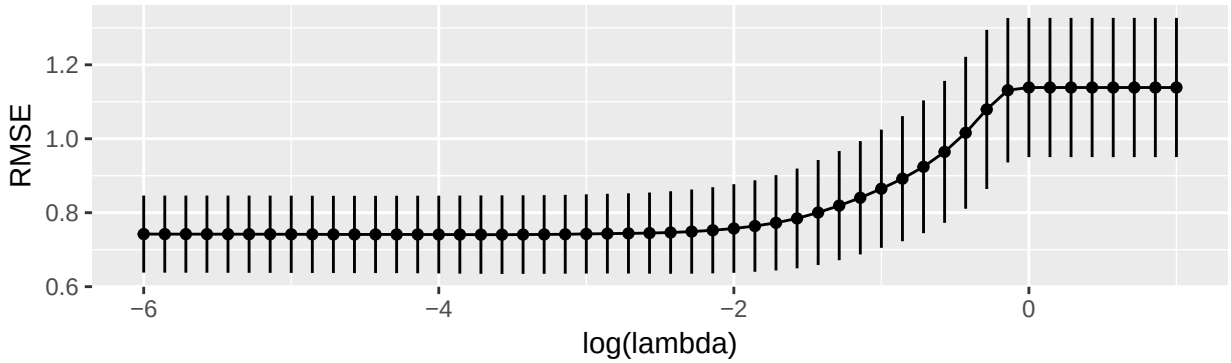



```
#autoplot(model$finalModel, xvar = 'lambda')   # ggplot version of this graph
```

Each line corresponds to the β_j coefficient for each λ value. The number at the top is the number of non-zero coefficients for that particular λ value.

Next we need to figure out the best value of λ that we considered.

```
# plot(model, xTrans = log, xlab='Log Lambda' )
ggplot(model$results, aes( x=log(lambda) )) +
  geom_point( aes(y=RMSE) ) +
  geom_line( aes(y=RMSE) ) +
  geom_linerange(aes( ymin= RMSE - RMSESD, ymax= RMSE+RMSESD))
```



So based on this graph, we want to choose λ to be as large a possible without increasing RMSE too much. So $\log(\lambda) \approx -2.4$ seems about right. And therefore $\lambda \approx e^{-2.4} = 0.025$

```
model$bestTune %>% data.frame() %>% mutate(log.Lambda=log(lambda))
```

```
##   alpha     lambda log.Lambda
## 1     1 0.02811566  -3.571429
```

So our best model contains 3 covariates

```
grid <- data.frame(
  alpha  = 1,  # 1 => Lasso Regression
  lambda = exp(c( -3.5, -3, -2, -1)))

model <- train( lpsa ~ ., data=prostate, method='glmnet',
                trControl=ctrl, tuneGrid=grid,
                lambda = grid$lambda )

model$finalModel$beta
```

```
## 8 x 4 sparse Matrix of class "dgCMatrix"
##                 s0           s1           s2           s3
## lcavol  0.38870682 0.4935910510  0.520590901  0.530253528
## lweight .          0.2681447342  0.361655752  0.396509232
## age     .          .            -0.002686174 -0.008525245
## lbph    .          0.0092857682  0.059409766  0.077594101
## svi     0.08924074 0.4551181593  0.579034073  0.603662364
## lcp     .          .             .            .
## gleason .          .             .            0.002218575
## pgg45   .          0.0001813713  0.001818373  0.002454284
```

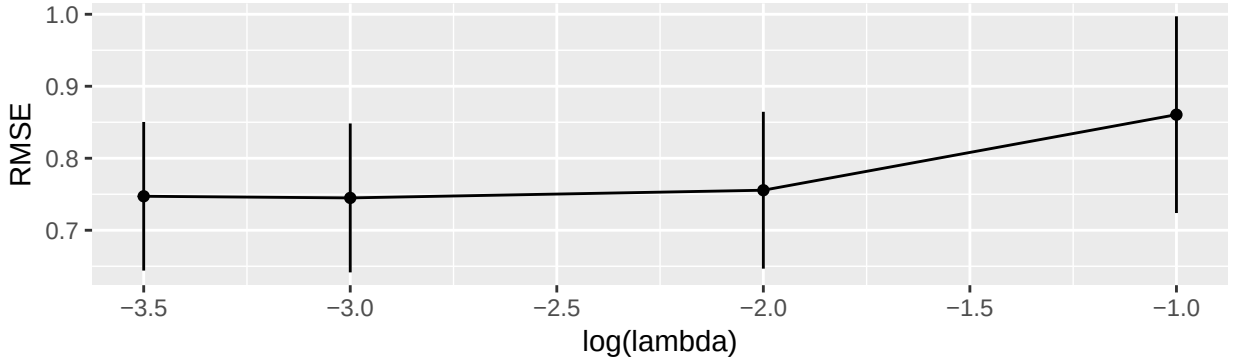
```
# This is the tune value that produced the smallest RMSE
model$finalModel$tuneValue %>% data.frame() %>%
  mutate( pow.lambda = 10^lambda)
```

```
##   alpha     lambda pow.lambda
## 1     1 0.04978707   1.121468
```

```
str( model$results )
```

```
## 'data.frame':    4 obs. of  8 variables:
##  $ alpha     : num  1 1 1 1
##  $ lambda    : num  0.0302 0.0498 0.1353 0.3679
##  $ RMSE      : num  0.747 0.745 0.756 0.861
##  $ Rsquared  : num  0.606 0.609 0.608 0.558
##  $ MAE       : num  0.589 0.591 0.601 0.658
##  $ RMSESD    : num  0.103 0.103 0.109 0.137
##  $ RsquaredSD: num  0.139 0.138 0.134 0.138
##  $ MAESD     : num  0.0896 0.088 0.0854 0.107
```

```
ggplot(model$results, aes( x=log(lambda) )) +
  geom_point( aes(y=RMSE) ) +
  geom_line( aes(y=RMSE) ) +
  geom_linerange(aes( ymin= RMSE - RMSESD, ymax= RMSE+RMSESD))
```



6.3 Dimension Reduction

Bibliography