STAT 206 Lecture

November 27, 2016

Lecture 1: Introduction: Basics of Data

Agenda

- Course overview
- Built-in data types
- Built-in functions and operators
- First data structures: Vectors and arrays

Why good statisticians learn to program?

- *Independence*: Otherwise, you rely on someone else having given you exactly the right tool
- *Honesty*: Otherwise, you end up distorting your problem to match the tools you have
- *Clarity*: Making your method something a machine can do disciplines your thinking and makes it public; that's science

How this class will work

- No programming knowledge presumed
- Some statistics knowledge presumed
- General programming mixed with data-manipulation and statistical inference
- Class will be *very* cumulative
- Keep up with the readings and assignments!
- Assignments, office hours, class notes, grading policies, useful links on R: http://www.faculty.ucr.edu/~jflegal

Overall class summary: Functional programming

2 sorts of things (**objects**): **data** and **functions**

- **Data**: things like 7, "seven", 7.000, the matrix $\begin{bmatrix} 7 & 7 & 7 \\ 7 & 7 & 7 \end{bmatrix}$
- **Functions**: things like log , + (two arguments), < (two), mod (two), `mean` (one)

  A function is a machine which turns input objects (**arguments**) into an output object (**return value**), possibly with **side effects**, according to a definite rule

Programming is writing functions to transform inputs into outputs

Good programming ensures the transformation is done easily and correctly

Machines are made out of machines; functions are made out of functions, like $f(a, b) = a^2 + b^2$

> The route to good programming is to take the big transformation and break it down into smaller ones, and then break those down, until you come to tasks which the built-in functions can do

Before functions, data

Different kinds of data object

All data is represented in binary format, by **bits** (TRUE/FALSE, YES/NO, 1/0)

- **Booleans** Direct binary values: `TRUE` or `FALSE` in R
- **Integers**: whole numbers (positive, negative or zero), represented by a fixed-length block of bits
- **Characters** fixed-length blocks of bits, with special coding; **strings** = sequences of characters
- **Floating point numbers**: a fraction (with a finite number of bits) times an exponent, like 1.87×10^6, but in binary form
- **Missing or ill-defined values**: `NA`, `NaN`, etc.

Operators

- **Unary** `-` for arithmetic negation, `!` for Boolean
- **Binary** usual arithmetic operators, plus ones for modulo and integer division; take two numbers and give a number

```
7 + 5
```

```
## [1] 12
```

```
7 - 5
```

```
## [1] 2
```

```
7 * 5
```

```
## [1] 35
```

```
7^5
```

```
## [1] 16807
```

```
7/5
```

```
## [1] 1.4
```

```
7%%5  # the modulo operator
```

```
## [1] 2
```

```
7%/%5  # indicates integer division
```

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

The R Console

Basic interaction with R is by typing in the **console**, a.k.a. **terminal** or **command-line**

You type in commands, R gives back answers (or errors)

Menus and other graphical interfaces are extras built on top of the console

Operators

Comparisons are also binary operators; they take two objects, like numbers, and give a Boolean

```
7 > 5
```

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

```
7 < 5
```

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

```
7 >= 7
```

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

```
7 <= 5
```

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

```
7 == 5
```

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

```
7 != 5
```

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

Boolean operators

Basically "and" and "or":

```
(5 > 7) & (6 * 7 == 42)
```

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

```
(5 > 7) | (6 * 7 == 42)
```

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

(will see special doubled forms, `&&` and `||`, later)

More types

`typeof()` function returns the type

`is.`*foo*`()` functions return Booleans for whether the argument is of type *foo*

`as.`*foo*`()` (tries to) "cast" its argument to type *foo* — to translate it sensibly into a *foo*-type value

```
typeof(7)
```

```
## [1] "double"
```

```
is.numeric(7)
```

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

```
is.na(7)
```

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

```
is.na(7/0)
```

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

```
is.na(0/0)
```

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

Why is 7/0 not NA, but 0/0 is?

```
is.character(7)
```

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

```
is.character("7")
```

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

```
is.character("seven")
```

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

```
is.na("seven")
```

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

```
as.character(5/6)
```

```
## [1] "0.833333333333333"
```

```
as.numeric(as.character(5/6))
```

```
## [1] 0.8333333
```

```
6 * as.numeric(as.character(5/6))
```

```
## [1] 5
```

```
5/6 == as.numeric(as.character(5/6))
```

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

(why is that last FALSE?)

Data can have names

Can give names to data objects; these give us **variables** (a few are built in)

```
pi
```

```
## [1] 3.141593
```

Variables can be arguments to functions or operators, just like constants:

```
pi * 10
```

```
## [1] 31.41593
```

```
cos(pi)
```

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

Most variables are created with the **assignment operator** <-

```
approx.pi <- 22/7
approx.pi
```

```
## [1] 3.142857
```

```
diameter.in.cubits = 10
approx.pi * diameter.in.cubits
```

```
## [1] 31.42857
```

The assignment operator also changes values:

```
circumference.in.cubits <- approx.pi * diameter.in.cubits
circumference.in.cubits
```

```
## [1] 31.42857
```

```
circumference.in.cubits <- 30
circumference.in.cubits
```

```
## [1] 30
```

Using names and variables makes code: easier to design, easier to debug, less prone to bugs, easier to improve, and easier for others to read

Avoid "magic constants"; use named variables you will be graded on this!

Named variables are a first step towards **abstraction**

The workspace

What names have you defined values for?

```
ls()
```

```
## [1] "approx.pi"               "circumference.in.cubits"
## [3] "diameter.in.cubits"
```

```
objects()
```

```
## [1] "approx.pi"               "circumference.in.cubits"
## [3] "diameter.in.cubits"
```

```
rm("circumference.in.cubits")
ls()
```

```
## [1] "approx.pi"          "diameter.in.cubits"
```

First data structure: vectors

Group related data values into one object, a **data structure**

A **vector** is a sequence of values, all of the same type

```
x <- c(7, 8, 10, 45)
x
```

```
## [1]  7  8 10 45
```

```
is.vector(x)
```

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

`c()` function returns a vector containing all its arguments in order

`x[1]` is the first element, `x[4]` is the 4th element

`x[-4]` is a vector containing all but the fourth element

```
x
```

```
## [1]  7  8 10 45
```

```
x[1]
```

```
## [1] 7
```

```
x[-4]
```

```
## [1]  7  8 10
```

`vector(length=6)` returns an empty vector of length 6; helpful for filling things up later

```
weekly.hours <- vector(length = 5)
weekly.hours[5] <- 8
```

Vector arithmetic

Operators apply to vectors "pairwise" or "elementwise":

```
y <- c(-7, -8, -10, -45)
x + y
```

```
## [1] 0 0 0 0
```

```
x * y
```

```
## [1]   -49   -64  -100 -2025
```

Recycling

Recycling repeat elements in shorter vector when combined with longer

```
x + c(-7, -8)
```

```
## [1]  0  0  3 37
```

```
x^c(1, 0, -1, 0.5)
```

```
## [1] 7.000000 1.000000 0.100000 6.708204
```

Single numbers are vectors of length 1 for purposes of recycling:

```
2 * x
```

```
## [1] 14 16 20 90
```

Can also do pairwise comparisons:

```
x > 9
```

```
## [1] FALSE FALSE  TRUE  TRUE
```

Note: returns Boolean vector

Boolean operators work elementwise:

```
(x > 9) & (x < 20)
```

```
## [1] FALSE FALSE  TRUE FALSE
```

To compare whole vectors, best to use `identical()` or `all.equal()`:

```
x == -y
```

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

```
identical(x, -y)
```

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

```
identical(c(0.5 - 0.3, 0.3 - 0.1), c(0.3 - 0.1, 0.5 - 0.3))
```

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

```
all.equal(c(0.5 - 0.3, 0.3 - 0.1), c(0.3 - 0.1, 0.5 - 0.3))
```

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

Functions on vectors

Lots of functions take vectors as arguments:

- `mean()`, `median()`, `sd()`, `var()`, `max()`, `min()`, `length()`, `sum()`: return single numbers
- `sort()` returns a new vector
- `hist()` takes a vector of numbers and produces a histogram, a highly structured object, with the side-effect of making a plot
- Similarly `ecdf()` produces a cumulative-density-function object
- `summary()` gives a five-number summary of numerical vectors
- `any()` and `all()` are useful on Boolean vectors

Addressing vectors

Vector of indices:

```
x[c(2, 4)]
```

```
## [1]  8 45
```

Vector of negative indices

```
x[c(-1, -3)]
```

```
## [1]  8 45
```

(why that, and not `8 10`?)

Boolean vector:

```
x[x > 9]
```

```
## [1] 10 45
```

```
y[x > 9]
```

```
## [1] -10 -45
```

`which()` turns a Boolean vector in vector of TRUE indices:

```
places <- which(x > 9)
places
```

```
## [1] 3 4
```

```
y[places]
```

```
## [1] -10 -45
```

Named components

You can give names to elements or components of vectors

```
names(x) <- c("v1", "v2", "v3", "fred")
names(x)
```

```
## [1] "v1"   "v2"   "v3"   "fred"
```

```
x[c("fred", "v1")]
```

```
## fred   v1
##   45    7
```

note the labels in what R prints; not actually part of the value

`names(x)` is just another vector (of characters):

```
names(y) <- names(x)
sort(names(x))
```

```
## [1] "fred" "v1"   "v2"   "v3"
```

```
which(names(x) == "fred")
```

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

Take-Aways

- We write programs by composing functions to manipulate data
- The basic data types let us represent Booleans, numbers, and characters
- Data structure let us group related values together
- Vectors let us group values of the same type
- Use variables rather a profusion of magic constants
- Name components of structures to make data more meaningful

Peculiarites of floating-point numbers

The more bits in the fraction part, the more precision

The R floating-point data type is a `double`, a.k.a. `numeric` back when memory was expensive, the now-standard number of bits was twice the default

Finite precision $\Rightarrow$ arithmetic on `doubles` $\neq$ arithmetic on $\mathbb{R}$.

```
0.45 == 3 * 0.15
```

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

```
0.45 - 3 * 0.15
```

```
## [1] 5.551115e-17
```

Often ignorable, but not always - Rounding errors tend to accumulate in long calculations - When results should be ≈ 0, errors can flip signs - Usually better to use `all.equal()` than exact comparison

```
(0.5 - 0.3) == (0.3 - 0.1)
```

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

```
all.equal(0.5 - 0.3, 0.3 - 0.1)
```

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

Peculiarities of Integers

Typing a whole number in the terminal doesn't make an integer; it makes a double, whose fractional part is 0

```
is.integer(7)
```

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

This looks like an integer

```
as.integer(7)
```

```
## [1] 7
```

To test for being a whole number, use `round()`:

```
round(7) == 7
```

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

Lecture 2: More Data Structures

Agenda

- Arrays
- Matrices
- Lists
- Dataframes
- Structures of structures

Arrays

Many data structures in R are made by adding bells and whistles to vectors, so "vector structures"

Most useful: **arrays**

```
x <- c(7, 8, 10, 45)
x.arr <- array(x, dim = c(2, 2))
x.arr
```

```
##      [,1] [,2]
## [1,]    7   10
## [2,]    8   45
```

`dim` says how many rows and columns; filled by columns

Can have $3, 4, \ldots n$ dimensional arrays; `dim` is a length-n vector

Some properties of the array:

```
dim(x.arr)
```

```
## [1] 2 2
```

```
is.vector(x.arr)
```

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

```
is.array(x.arr)
```

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

```
typeof(x.arr)
```

```
## [1] "double"
```

```
str(x.arr)
```

```
##  num [1:2, 1:2] 7 8 10 45
```

```
attributes(x.arr)
```

```
## $dim
## [1] 2 2
```

`typeof()` returns the type of the *elements*

`str()` gives the **structure**: here, a numeric array, with two dimensions, both indexed 1–2, and then the actual numbers

Exercise: try all these with `x`

Accessing and operating on arrays

Can access a 2-D array either by pairs of indices or by the underlying vector:

```
x.arr[1, 2]
```

```
## [1] 10
```

```
x.arr[3]
```

```
## [1] 10
```

Omitting an index means "all of it":

```
x.arr[c(1:2), 2]
```

```
## [1] 10 45
```

```
x.arr[, 2]
```

```
## [1] 10 45
```

Functions on arrays

Using a vector-style function on a vector structure will go down to the underlying vector, *unless* the function is set up to handle arrays specially:

```
which(x.arr > 9)
```

```
## [1] 3 4
```

Many functions *do* preserve array structure:

```
y <- -x
y.arr <- array(y, dim = c(2, 2))
y.arr + x.arr
```

```
##      [,1] [,2]
## [1,]    0    0
## [2,]    0    0
```

Others specifically act on each row or column of the array separately:

```
rowSums(x.arr)
```

```
## [1] 17 53
```

We will see a lot more of this idea

Example: Price of houses in PA

Census data for California and Pennsylvania on housing prices, by Census "tract"

```
calif_penn <- read.csv("http://www.stat.cmu.edu/~cshalizi/uADA/13/hw/01/calif_penn_2011.csv")
penn <- calif_penn[calif_penn[, "STATEFP"] == 42, ]
coefficients(lm(Median_house_value ~ Median_household_income, data = penn))
```

```
##             (Intercept) Median_household_income
##           -26206.564325                3.651256
```

Fit a simple linear model, predicting median house price from median household income

Census tracts 24–425 are Allegheny county

Tract 24 has a median income of $14,719; actual median house value is $34,100 — is that above or below what's?

```
34100 < -26206.564 + 3.651 * 14719
```

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

Tract 25 has income $48,102 and house price $155,900

```
155900 < -26206.564 + 3.651 * 48102
```

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

What about tract 26?

We *could* just keep plugging in numbers like this, but that's

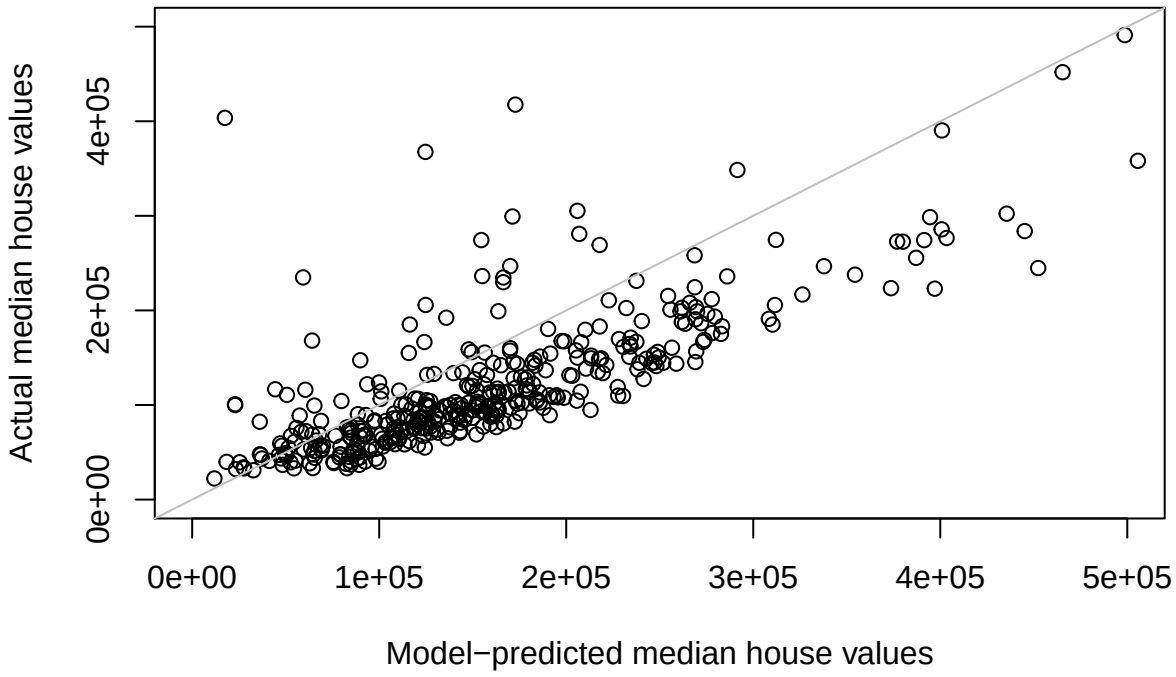
- boring and repetitive
- error-prone (what if I forget to change the median income, or drop a minus sign from the intercept?)
- obscure if we come back to our work later (what *are* these numbers?)

Use variables and names

```
penn.coefs <- coefficients(lm(Median_house_value ~ Median_household_income,
    data = penn))
penn.coefs
```

```
##             (Intercept) Median_household_income
##           -26206.564325                3.651256
```

```
allegheny.rows <- 24:425
allegheny.medinc <- penn[allegheny.rows, "Median_household_income"]
allegheny.values <- penn[allegheny.rows, "Median_house_value"]
allegheny.fitted <- penn.coefs["(Intercept)"] + penn.coefs["Median_household_income"] *
    allegheny.medinc
plot(x = allegheny.fitted, y = allegheny.values, xlab = "Model-predicted median house values",
    ylab = "Actual median house values", xlim = c(0, 5e+05), ylim = c(0, 5e+05))
abline(a = 0, b = 1, col = "grey")
```



Running example: resource allocation ("mathematical programming")

Factory makes cars and trucks, using labor and steel

- a car takes 40 hours of labor and 1 ton of steel
- a truck takes 60 hours and 3 tons of steel
- resources: 1600 hours of labor and 70 tons of steel each week

Matrices

In R, a matrix is a specialization of a 2D array

```
factory <- matrix(c(40, 1, 60, 3), nrow = 2)
is.array(factory)
```

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

```
is.matrix(factory)
```

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

could also specify `ncol`, and/or `byrow=TRUE` to fill by rows.

Element-wise operations with the usual arithmetic and comparison operators (e.g., `factory/3`)

Compare whole matrices with `identical()` or `all.equal()`

Matrix multiplication

Gets a special operator

```
six.sevens <- matrix(rep(7, 6), ncol = 3)
six.sevens
```

```
##      [,1] [,2] [,3]
## [1,]    7    7    7
## [2,]    7    7    7
```

```
factory %*% six.sevens  # [2x2] * [2x3]
```

```
##      [,1] [,2] [,3]
## [1,]  700  700  700
## [2,]   28   28   28
```

What happens if you try `six.sevens %*% factory`?

Multiplying matrices and vectors

Numeric vectors can act like proper vectors:

```
output <- c(10, 20)
factory %*% output
```

```
##      [,1]
## [1,] 1600
## [2,]   70
```

```
output %*% factory
```

```
##      [,1] [,2]
## [1,]  420  660
```

R silently casts the vector as either a row or a column matrix

Matrix operators

Transpose:

```
t(factory)
```

```
##      [,1] [,2]
## [1,]   40    1
## [2,]   60    3
```

Determinant:

```
det(factory)
```

```
## [1] 60
```

The diagonal

The `diag()` function can extract the diagonal entries of a matrix:

```
diag(factory)
```

```
## [1] 40  3
```

It can also *change* the diagonal:

```
diag(factory) <- c(35, 4)
factory
```

```
##      [,1] [,2]
## [1,]   35   60
## [2,]    1    4
```

Re-set it for later:

```
diag(factory) <- c(40, 3)
```

Creating a diagonal or identity matrix

```
diag(c(3, 4))
```

```
##      [,1] [,2]
## [1,]    3    0
## [2,]    0    4
```

```
diag(2)
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

Inverting a matrix

```
solve(factory)
```

```
##             [,1]       [,2]
## [1,]  0.05000000 -1.0000000
## [2,] -0.01666667  0.6666667
```

```
factory %*% solve(factory)
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

Why's it called "solve"" anyway?

Solving the linear system $\mathbf{A}\vec{x} = \vec{b}$ for $\vec{x}$:

```
available <- c(1600, 70)
solve(factory, available)
```

```
## [1] 10 20
```

```
factory %*% solve(factory, available)
```

```
##      [,1]
## [1,] 1600
## [2,]   70
```

Names in matrices

We can name either rows or columns or both, with `rownames()` and `colnames()`

These are just character vectors, and we use the same function to get and to set their values

Names help us understand what we're working with

Names can be used to coordinate different objects

```
rownames(factory) <- c("labor", "steel")
colnames(factory) <- c("cars", "trucks")
factory
```

```
##       cars trucks
## labor   40     60
## steel    1      3
```

```
available <- c(1600, 70)
names(available) <- c("labor", "steel")
output <- c(20, 10)
names(output) <- c("trucks", "cars")
factory %*% output  # But we've got cars and trucks mixed up!
```

```
##       [,1]
## labor 1400
## steel   50
```

```
factory %*% output[colnames(factory)]
```

```
##       [,1]
## labor 1600
## steel   70
```

```
all(factory %*% output[colnames(factory)] <= available[rownames(factory)])
```

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

Notice: Last lines don't have to change if we add motorcycles as output or rubber and glass as inputs (abstraction again)

Doing the same thing to each row or column

Take the mean: `rowMeans()`, `colMeans()`: input is matrix, output is vector. Also `rowSums()`, etc.

`summary()`: vector-style summary of column

```
colMeans(factory)
```

```
##   cars trucks
##   20.5   31.5
```

```
summary(factory)
```

```
##       cars           trucks
##  Min.   : 1.00   Min.   : 3.00
##  1st Qu.:10.75   1st Qu.:17.25
##  Median :20.50   Median :31.50
##  Mean   :20.50   Mean   :31.50
##  3rd Qu.:30.25   3rd Qu.:45.75
##  Max.   :40.00   Max.   :60.00
```

`apply()`, takes 3 arguments: the array or matrix, then 1 for rows and 2 for columns, then name of the function to apply to each

```
rowMeans(factory)
```

```
## labor steel
##    50     2
```

```
apply(factory, 1, mean)
```

```
## labor steel
##    50     2
```

What would `apply(factory,1,sd)` do?

Lists

Sequence of values, *not* necessarily all of the same type

```
my.distribution <- list("exponential", 7, FALSE)
my.distribution
```

```
## [[1]]
## [1] "exponential"
##
## [[2]]
## [1] 7
##
## [[3]]
## [1] FALSE
```

Most of what you can do with vectors you can also do with lists

Accessing pieces of lists

Can use `[ ]` as with vectors
or use `[[ ]]`, but only with a single index
`[[ ]]` drops names and structures, `[ ]` does not

```
is.character(my.distribution)
```

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

```
is.character(my.distribution[[1]])
```

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

```
my.distribution[[2]]^2
```

```
## [1] 49
```

What happens if you try `my.distribution[2]^2`? What happens if you try `[[ ]]` on a vector?

Expanding and contracting lists

Add to lists with `c()` (also works with vectors):

```
my.distribution <- c(my.distribution, 7)
my.distribution
```

```
## [[1]]
## [1] "exponential"
##
## [[2]]
## [1] 7
##
## [[3]]
## [1] FALSE
##
## [[4]]
## [1] 7
```

Chop off the end of a list by setting the length to something smaller (also works with vectors):

```
length(my.distribution)
```

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

```
length(my.distribution) <- 3
my.distribution
```

```
## [[1]]
## [1] "exponential"
##
## [[2]]
## [1] 7
##
## [[3]]
## [1] FALSE
```

Naming list elements

We can name some or all of the elements of a list

```
names(my.distribution) <- c("family", "mean", "is.symmetric")
my.distribution
```

```
## $family
## [1] "exponential"
##
## $mean
## [1] 7
##
## $is.symmetric
## [1] FALSE
```

```
my.distribution[["family"]]
```

```
## [1] "exponential"
```

```
my.distribution["family"]
```

```
## $family
## [1] "exponential"
```

Lists have a special short-cut way of using names, `$` (which removes names and structures):

```
my.distribution[["family"]]
```

```
## [1] "exponential"
```

```
my.distribution$family
```

```
## [1] "exponential"
```

Names in lists

Creating a list with names:

```
another.distribution <- list(family = "gaussian", mean = 7, sd = 1, is.symmetric = TRUE)
```

Adding named elements:

```
my.distribution$was.estimated <- FALSE
my.distribution[["last.updated"]] <- "2011-08-30"
```

Removing a named list element, by assigning it the value `NULL`:

```
my.distribution$was.estimated <- NULL
```

Key-Value pairs

Lists give us a way to store and look up data by *name*, rather than by *position*

A really useful programming concept with many names: **key-value pairs**, **dictionaries**, **associative arrays**, **hashes**

If all our distributions have components named `family`, we can look that up by name, without caring where it is in the list

Dataframes

Dataframe = the classic data table, n rows for cases, p columns for variables

Lots of the really-statistical parts of R presume data frames `penn` from last time was really a dataframe

Not just a matrix because *columns can have different types*

Many matrix functions also work for dataframes (`rowSums()`, `summary()`, `apply()`)

but no matrix multiplying dataframes, even if all columns are numeric

```
a.matrix <- matrix(c(35, 8, 10, 4), nrow = 2)
colnames(a.matrix) <- c("v1", "v2")
a.matrix
```

```
##      v1 v2
## [1,] 35 10
## [2,]  8  4
```

```
a.matrix[, "v1"]  # Try a.matrix$v1 and see what happens
```

```
## [1] 35  8
```

```
a.data.frame <- data.frame(a.matrix, logicals = c(TRUE, FALSE))
a.data.frame
```

```
##   v1 v2 logicals
## 1 35 10     TRUE
## 2  8  4    FALSE
```

```
a.data.frame$v1
```

```
## [1] 35  8
```

```
a.data.frame[, "v1"]
```

```
## [1] 35  8
```

```
a.data.frame[1, ]
```

```
##   v1 v2 logicals
## 1 35 10     TRUE
```

```
colMeans(a.data.frame)
```

```
##       v1       v2 logicals
##     21.5      7.0      0.5
```

Adding rows and columns

We can add rows or columns to an array or data-frame with `rbind()` and `cbind()`, but be careful about forced type conversions

```
rbind(a.data.frame, list(v1 = -3, v2 = -5, logicals = TRUE))
```

```
##   v1 v2 logicals
## 1 35 10     TRUE
## 2  8  4    FALSE
## 3 -3 -5     TRUE
```

```
rbind(a.data.frame, c(3, 4, 6))
```

```
##   v1 v2 logicals
## 1 35 10        1
## 2  8  4        0
## 3  3  4        6
```

Structures of Structures

So far, every list element has been a single data value

List elements can be other data structures, e.g., vectors and matrices:

```
plan <- list(factory = factory, available = available, output = output)
plan$output
```

```
## trucks   cars
##     20     10
```

Internally, a dataframe is basically a list of vectors

Structures of Structures

List elements can even be other lists
which may contain other data structures
including other lists
which may contain other data structures...

This **recursion** lets us build arbitrarily complicated data structures from the basic ones

Most complicated objects are (usually) lists of data structures

Example: Eigenstuff

`eigen()` finds eigenvalues and eigenvectors of a matrix
Returns a list of a vector (the eigenvalues) and a matrix (the eigenvectors)

```
eigen(factory)
```

```
## $values
## [1] 41.556171  1.443829
##
## $vectors
##            [,1]       [,2]
## [1,] 0.99966383 -0.8412758
## [2,] 0.02592747  0.5406062
```

```
class(eigen(factory))
```

```
## [1] "list"
```

With complicated objects, you can access parts of parts (of parts...)

```
factory %*% eigen(factory)$vectors[, 2]
```

```
##             [,1]
## labor -1.2146583
## steel  0.7805429
```

```
eigen(factory)$values[2] * eigen(factory)$vectors[, 2]
```

```
## [1] -1.2146583  0.7805429
```

```
eigen(factory)$values[2]
```

```
## [1] 1.443829
```

```
eigen(factory)[[1]][[2]]  # NOT [[1,2]]
```

```
## [1] 1.443829
```

Creating an example dataframe

```
library(datasets)
states <- data.frame(state.x77, abb = state.abb, region = state.region, division = state.division)
```

`data.frame()` is combining here a pre-existing matrix (`state.x77`), a vector of characters (`state.abb`), and two vectors of qualitative categorical variables (**factors**; `state.region`, `state.division`)

Column names are preserved or guessed if not explicitly set

```
colnames(states)
```

```
##  [1] "Population" "Income"     "Illiteracy" "Life.Exp"   "Murder"
##  [6] "HS.Grad"    "Frost"      "Area"       "abb"        "region"
## [11] "division"
```

```
states[1, ]
```

```
##         Population Income Illiteracy Life.Exp Murder HS.Grad Frost  Area
## Alabama       3615   3624        2.1    69.05   15.1    41.3    20 50708
##         abb region           division
## Alabama  AL  South East South Central
```

Dataframe access

- By row and column index

```
states[49, 3]
```

```
## [1] 0.7
```

- By row and column names

```
states["Wisconsin", "Illiteracy"]
```

```
## [1] 0.7
```

Dataframe access

- All of a row:

```
states["Wisconsin", ]
```

```
##           Population Income Illiteracy Life.Exp Murder HS.Grad Frost  Area
## Wisconsin       4589   4468        0.7    72.48      3    54.5   149 54464
##           abb        region           division
## Wisconsin  WI North Central East North Central
```

Exercise: what class is `states["Wisconsin",]`?

- All of a column:

```
head(states[, 3])
```

```
## [1] 2.1 1.5 1.8 1.9 1.1 0.7
```

```
head(states[, "Illiteracy"])
```

```
## [1] 2.1 1.5 1.8 1.9 1.1 0.7
```

```
head(states$Illiteracy)
```

```
## [1] 2.1 1.5 1.8 1.9 1.1 0.7
```

- Rows matching a condition:

```
states[states$division == "New England", "Illiteracy"]
```

```
## [1] 1.1 0.7 1.1 0.7 1.3 0.6
```

```
states[states$region == "South", "Illiteracy"]
```

```
##  [1] 2.1 1.9 0.9 1.3 2.0 1.6 2.8 0.9 2.4 1.8 1.1 2.3 1.7 2.2 1.4 1.4
```

Parts or all of the dataframe can be assigned to:

```
summary(states$HS.Grad)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   37.80   48.05   53.25   53.11   59.15   67.30
```

```
states$HS.Grad <- states$HS.Grad/100
summary(states$HS.Grad)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.3780  0.4805  0.5325  0.5311  0.5915  0.6730
```

```
states$HS.Grad <- 100 * states$HS.Grad
```

What percentage of literate adults graduated HS?

```
head(100 * (states$HS.Grad/(100 - states$Illiteracy)))
```

```
## [1] 42.18590 67.71574 59.16497 40.67278 63.29626 64.35045
```

`with()` takes a data frame and evaluates an expression "inside" it:

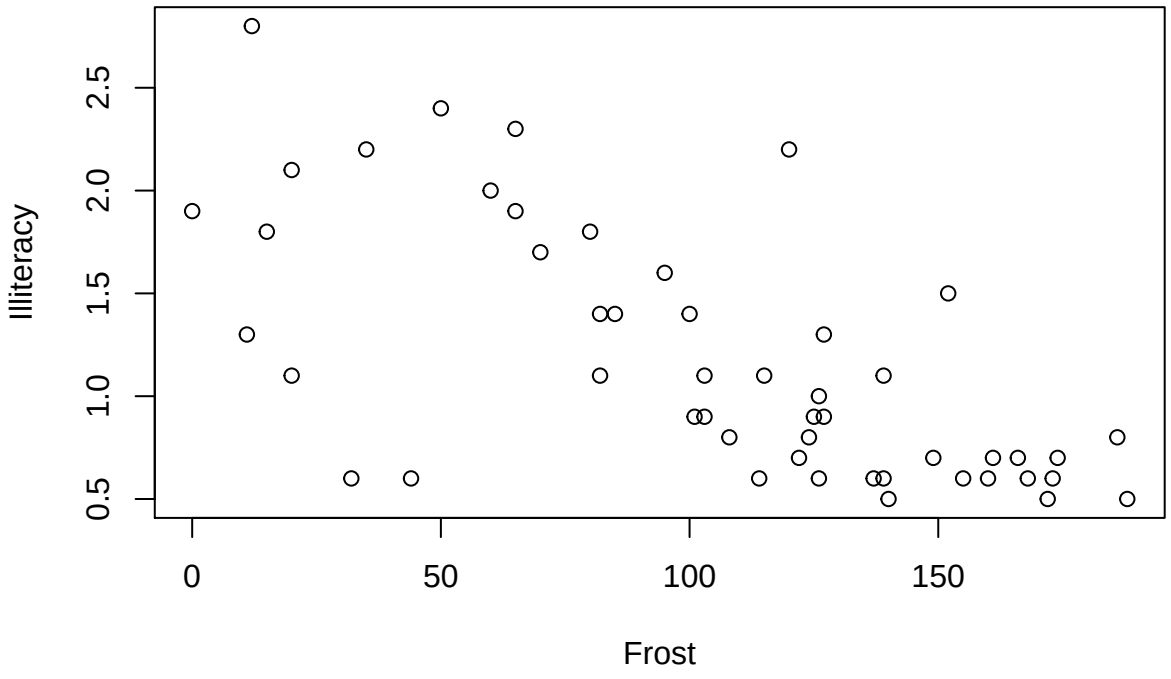
```
with(states, head(100 * (HS.Grad/(100 - Illiteracy))))
```

```
## [1] 42.18590 67.71574 59.16497 40.67278 63.29626 64.35045
```

Data arguments

Lots of functions take `data` arguments, and look variables up in that data frame:

```
plot(Illiteracy ~ Frost, data = states)
```



$R^2 = 0.45$, $p \approx 10^{-7}$

Summary

- Arrays add multi-dimensional structure to vectors
- Matrices act like you'd hope they would
- Lists let us combine different types of data
- Dataframes are hybrids of matrices and lists, for classic tabular data
- Recursion lets us build complicated data structures out of the simpler ones

Lecture 3: Control Flow

Agenda

- Control flow (or alternatively, flow of control)
- if(), for(), and while()
- Avoiding iteration
- Introduction to strings and string operations

Control flow

Control flow is the order in which individual statements, instructions or function calls of an imperative program are executed or evaluated

A *control flow statement* is a statement whose execution results in a choice being made as to which of two or more paths should be followed

Conditionals

Have the computer decide what to do next

- Mathematically:

$$|x| = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0 \end{cases}, \quad \psi(x) = \begin{cases} x^2 & \text{if } |x| \leq 1 \\ 2|x| - 1 & \text{if } |x| > 1 \end{cases}$$

  Exercise: plot ψ in R

- Computationally:

```
if the country code is not "US", multiply prices by current exchange rate
```

If Statements

Simplest conditional:

```
if (x >= 0) {
  x
} else {
  -x
}
```

Condition in `if` needs to give *one* `TRUE` or `FALSE` value

`else` clause is optional

one-line actions don't need braces

```
if (x >= 0) x else -x
```

`if` can *nest* arbitrarily deeply:

```
if (x^2 < 1) {
  x^2
} else {
  if (x >= 0) {
    2*x-1
```

```
  } else {
     -2*x-1
  }
}
```

Can get ugly though

Combining Booleans: && and ||

`&` work `|` like `+` or `*`: combine terms element-wise

Flow control wants *one* Boolean value, and to skip calculating what's not needed

`&&` and `||` give *one* Boolean, lazily:

```
(0 > 0) && (all.equal(42%%6, 169%%13))
```

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

This *never* evaluates the complex expression on the right

Use `&&` and `||` for control, `&` and `|` for subsetting

Iteration

Repeat similar actions multiple times:

```
table.of.logarithms <- vector(length = 7, mode = "numeric")
table.of.logarithms
```

```
## [1] 0 0 0 0 0 0 0
```

```
for (i in 1:length(table.of.logarithms)) {
    table.of.logarithms[i] <- log(i)
}
table.of.logarithms
```

```
## [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
```

For Loops

```
for (i in 1:length(table.of.logarithms)) {
  table.of.logarithms[i] <- log(i)
}
```

`for` increments a **counter** (here `i`) along a vector (here `1:length(table.of.logarithms)`) and **loops through** the **body* until it runs through the vector

"**iterates over** the vector"

Note, there is a better way to do this job!

Can contain just about anything, including:

- if() clauses
- other for() loops (nested iteration)

Nested iteration example

```
c <- matrix(0, nrow=nrow(a), ncol=ncol(b))
if (ncol(a) == nrow(b)) {
  for (i in 1:nrow(c)) {
    for (j in 1:ncol(c)) {
      for (k in 1:ncol(a)) {
        c[i,j] <- c[i,j] + a[i,k]*b[k,j]
      }
    }
  }
} else {
  stop("matrices a and b non-conformable")
}
```

While Loops

```
while (max(x) - 1 > 1e-06) {
  x <- sqrt(x)
}
```

Condition in the argument to `while` must be a single Boolean value (like `if`)

Body is looped over until the condition is `FALSE` so can loop forever

Loop never begins unless the condition starts `TRUE`

for() vs. while()

for() is better when the number of times to repeat (values to iterate over) is clear in advance

while() is better when you can recognize when to stop once you're there, even if you can't guess it to begin with

Every for() could be replaced with a while()
Exercise: show this

Avoiding iteration

R has many ways of *avoiding* iteration, by acting on whole objects

- It's conceptually clearer
- It leads to simpler code
- It's faster (sometimes a little, sometimes drastically)

Vectorized arithmetic

How many languages add 2 vectors:

```
c <- vector(length(a))
for (i in 1:length(a)) {  c[i] <- a[i] + b[i]  }
```

How R adds 2 vectors:

```
a+b
```

or a triple `for()` loop for matrix multiplication vs. `a %*% b`

Advantages of vectorizing

- Clarity: the syntax is about *what* we're doing
- Concision: we write less
- Abstraction: the syntax hides *how the computer does it*
- Generality: same syntax works for numbers, vectors, arrays, ...
- Speed: modifying big vectors over and over is slow in R; work gets done by optimized low-level code

Vectorized calculations

Many functions are set up to vectorize automatically

```
abs(-3:3)
```

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

```
log(1:7)
```

```
## [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
```

See also `apply()`

Vectorized conditions: ifelse()

```
ifelse(x^2 > 1, 2*abs(x)-1, x^2)
```

1st argument is a Boolean vector, then pick from the 2nd or 3rd vector arguments as `TRUE` or `FALSE`

What Is Truth?

0 counts as `FALSE`; other numeric values count as `TRUE`; the strings "TRUE" and "FALSE" count as you'd hope; most everything else gives an error

Advice: Don't play games here; try to make sure control expressions are getting Boolean values

Conversely, in arithmetic, `FALSE` is 0 and `TRUE` is 1

```
library(datasets)
states <- data.frame(state.x77, abb = state.abb, region = state.region, division = state.division)
mean(states$Murder > 7)
```

```
## [1] 0.48
```

Switch Function

Simplify nested `if` with `switch()`: give a variable to select on, then a value for each option

```
switch(type.of.summary,
       mean=mean(states$Murder),
       median=median(states$Murder),
       histogram=hist(states$Murder),
       "I don't understand")
```

Exercise (off-line)

Set `type.of.summary` to, succesively, "mean", "median", "histogram", and "mode", and explain what happens

Unconditional iteration

```
repeat {
  print("Help! I am Dr. Morris Culpepper, trapped in an endless loop!")
}
```

"Manual" control over iteration

```
repeat {
  if (watched) { next() }
  print("Help! I am Dr. Morris Culpepper, trapped in an endless loop!")
  if (rescued) { break() }
}
```

`break()` exits the loop; `next()` skips the rest of the body and goes back into the loop

both work with `for()` and `while()` as well

Exercise: how would you replace `while()` with `repeat()`?

Strings and string operations

Most data we deal with is in character form!

- web pages can be scraped
- email can be analyzed for network properties
- survey responses must be processed and compared

Even if you only care about numbers, it helps to be able to extract them from text and manipulate them easily.

Characters vs. Strings

- ***Character***: a symbol in a written language, specifically what you can enter at a keyboard: letters, numerals, punctuation, space, newlines, etc.

`'L', 'i', 'n', 'c', 'o', 'l'`

- ***String***: a sequence of characters bound together

  `Lincoln`

Note: R does not have a separate type for characters and strings

```
mode("L")
```

```
## [1] "character"
```

```
mode("Lincoln")
```

```
## [1] "character"
```

```
class("Lincoln")
```

```
## [1] "character"
```

Making Strings

Use single or double quotes to construct a string; use `nchar()` to get the length of a single string. Why do we prefer double quotes?

```
"Lincoln"
```

```
## [1] "Lincoln"
```

```
"Abraham Lincoln"
```

```
## [1] "Abraham Lincoln"
```

```
"Abraham Lincoln's Hat"
```

```
## [1] "Abraham Lincoln's Hat"
```

```
"As Lincoln never said, \"Four score and seven beers ago\""
```

```
## [1] "As Lincoln never said, \"Four score and seven beers ago\""
```

Whitespace

The space, `" "` is a character; so are multiple spaces `"  "` and the empty string, `""`.

Some characters are special, so we have "escape characters" to specify them in strings. - quotes within strings: `\"` - tab: `\t` - new line `\n` and carriage return `\r` – use the former rather than the latter when possible

Character data type

One of the atomic data types, like `numeric` or `logical`

Can go into scalars, vectors, arrays, lists, or be the type of a column in a data frame.

```
length("Abraham Lincoln's beard")
```

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

```
length(c("Abraham", "Lincoln's", "beard"))
```

```
## [1] 3
```

```
nchar("Abraham Lincoln's beard")
```

```
## [1] 23
```

```
nchar(c("Abraham", "Lincoln's", "beard"))
```

```
## [1] 7 9 5
```

Character-valued variables

They work just like others, e.g., with vectors:

```
president <- "Lincoln"
nchar(president)  # NOT 9
```

```
## [1] 7
```

```
presidents <- c("Fillmore", "Pierce", "Buchanan", "Davis", "Johnson")
presidents[3]
```

```
## [1] "Buchanan"
```

```
presidents[-(1:3)]
```

```
## [1] "Davis"   "Johnson"
```

Displaying characters

We know `print()`, of course; `cat()` writes the string directly to the console. If you're debugging, `message()` is R's preferred syntax.

```
print("Abraham Lincoln")
```

```
## [1] "Abraham Lincoln"
```

```
cat("Abraham Lincoln")
```

```
## Abraham Lincoln
```

```
cat(presidents)
```

```
## Fillmore Pierce Buchanan Davis Johnson
```

```
message(presidents)
```

```
## FillmorePierceBuchananDavisJohnson
```

Substring operations

Substring: a smaller string from the big string, but still a string in its own right.

A string is not a vector or a list, so we ***cannot*** use subscripts like `[[ ]]` or `[ ]` to extract substrings; we use `substr()` instead.

```
phrase <- "Christmas Bonus"
substr(phrase, start = 8, stop = 12)
```

```
## [1] "as Bo"
```

We can also use `substr` to replace elements:

```
substr(phrase, 13, 13) <- "g"
phrase
```

```
## [1] "Christmas Bogus"
```

substr() for string vectors

`substr()` vectorizes over all its arguments:

```
presidents
```

```
## [1] "Fillmore" "Pierce"   "Buchanan" "Davis"    "Johnson"
```

```
substr(presidents, 1, 2)  # First two characters
```

```
## [1] "Fi" "Pi" "Bu" "Da" "Jo"
```

```
substr(presidents, nchar(presidents) - 1, nchar(presidents))  # Last two
```

```
## [1] "re" "ce" "an" "is" "on"
```

```
substr(presidents, 20, 21)  # No such substrings so return the null string
```

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

```
substr(presidents, 7, 7)  # Explain!
```

```
## [1] "r" ""  "a" ""  "n"
```

Dividing strings into vectors

`strsplit()` divides a string according to key characters, by splitting each element of the character vector `x` at appearances of the pattern `split`.

```
scarborough.fair <- "parsley, sage, rosemary, thyme"
strsplit(scarborough.fair, ",")
```

```
## [[1]]
## [1] "parsley"   " sage"     " rosemary" " thyme"
```

```
strsplit(scarborough.fair, ", ")
```

```
## [[1]]
## [1] "parsley"  "sage"     "rosemary" "thyme"
```

Pattern is recycled over elements of the input vector:

```
strsplit(c(scarborough.fair, "Garfunkel, Oates", "Clement, McKenzie"), ", ")
```

```
## [[1]]
## [1] "parsley"  "sage"     "rosemary" "thyme"
##
## [[2]]
## [1] "Garfunkel" "Oates"
##
## [[3]]
## [1] "Clement"  "McKenzie"
```

Note that it outputs a `list` of character vectors – why should this be the default?

Combining vectors into strings

Converting one variable type to another is called ***casting***:

```
as.character(7.2)  # Obvious
```

```
## [1] "7.2"
```

```r
as.character(7.2e+12)  # Obvious
```

```
## [1] "7.2e+12"
```

```r
as.character(c(7.2, 7.2e+12))  # Obvious
```

```
## [1] "7.2"     "7.2e+12"
```

```r
as.character(720000)  # Not quite so obvious
```

```
## [1] "720000"
```

Building strings from multiple parts

The `paste()` function is very flexible!

With one vector argument, works like `as.character()`:

```r
paste(41:45)
```

```
## [1] "41" "42" "43" "44" "45"
```

With 2 or more vector arguments, combines them with recycling:

```r
paste(presidents, 41:45)
```

```
## [1] "Fillmore 41" "Pierce 42"   "Buchanan 43" "Davis 44"    "Johnson 45"
```

```r
paste(presidents, c("R", "D"))  # Not historically accurate!
```

```
## [1] "Fillmore R" "Pierce D"   "Buchanan R" "Davis D"    "Johnson R"
```

```r
paste(presidents, "(", c("R", "D"), 41:45, ")")
```

```
## [1] "Fillmore ( R 41 )" "Pierce ( D 42 )"   "Buchanan ( R 43 )"
## [4] "Davis ( D 44 )"    "Johnson ( R 45 )"
```

Changing the separator between pasted-together terms:

```r
paste(presidents, " (", 41:45, ")", sep = "_")
```

```
## [1] "Fillmore_ (_41_)" "Pierce_ (_42_)"   "Buchanan_ (_43_)"
## [4] "Davis_ (_44_)"    "Johnson_ (_45_)"
```

```r
paste(presidents, " (", 41:45, ")", sep = "")
```

```
## [1] "Fillmore (41)" "Pierce (42)"   "Buchanan (43)" "Davis (44)"
## [5] "Johnson (45)"
```

Exercise: what happens if you give `sep` a vector?

More complicated example of recycling

Exercise: Convince yourself of why this works as it does

```r
paste(c("HW", "Lab"), rep(1:11, times = rep(2, 11)))
```

```
##  [1] "HW 1"   "Lab 1"  "HW 2"   "Lab 2"  "HW 3"   "Lab 3"  "HW 4"
##  [8] "Lab 4"  "HW 5"   "Lab 5"  "HW 6"   "Lab 6"  "HW 7"   "Lab 7"
## [15] "HW 8"   "Lab 8"  "HW 9"   "Lab 9"  "HW 10"  "Lab 10" "HW 11"
## [22] "Lab 11"
```

Condensing multiple strings

Producing one big string:

```
paste(presidents, " (", 41:45, ")", sep = "", collapse = "; ")
```

```
## [1] "Fillmore (41); Pierce (42); Buchanan (43); Davis (44); Johnson (45)"
```

Default value of `collapse` is `NULL` – that is, it won't use it

Function for writing regression formulas

R has a standard syntax for models: outcome and predictors.

```
my.formula <- function(dep, indeps, df) {
    rhs <- paste(colnames(df)[indeps], collapse = "+")
    return(paste(colnames(df)[dep], " ~  ", rhs, collapse = ""))
}
my.formula(2, c(3, 5, 7), df = state.x77)
```

```
## [1] "Income  ~  Illiteracy+Murder+Frost"
```

General search

- Use `grep()` to find which strings have a matching search term
- Reconstituting, make one long string, then split the words
- Counting words with `table()`
- Need to learn how to work with text patterns and not just constants
- Searching for text patterns using regular expressions

Summary

- `if`, nested `if`, `switch`
- Iteration: `for`, `while`
- Avoiding iteration with whole-object ("vectorized") operations
- Text is data, just like everything else

Lecture 4: Graphics

Agenda

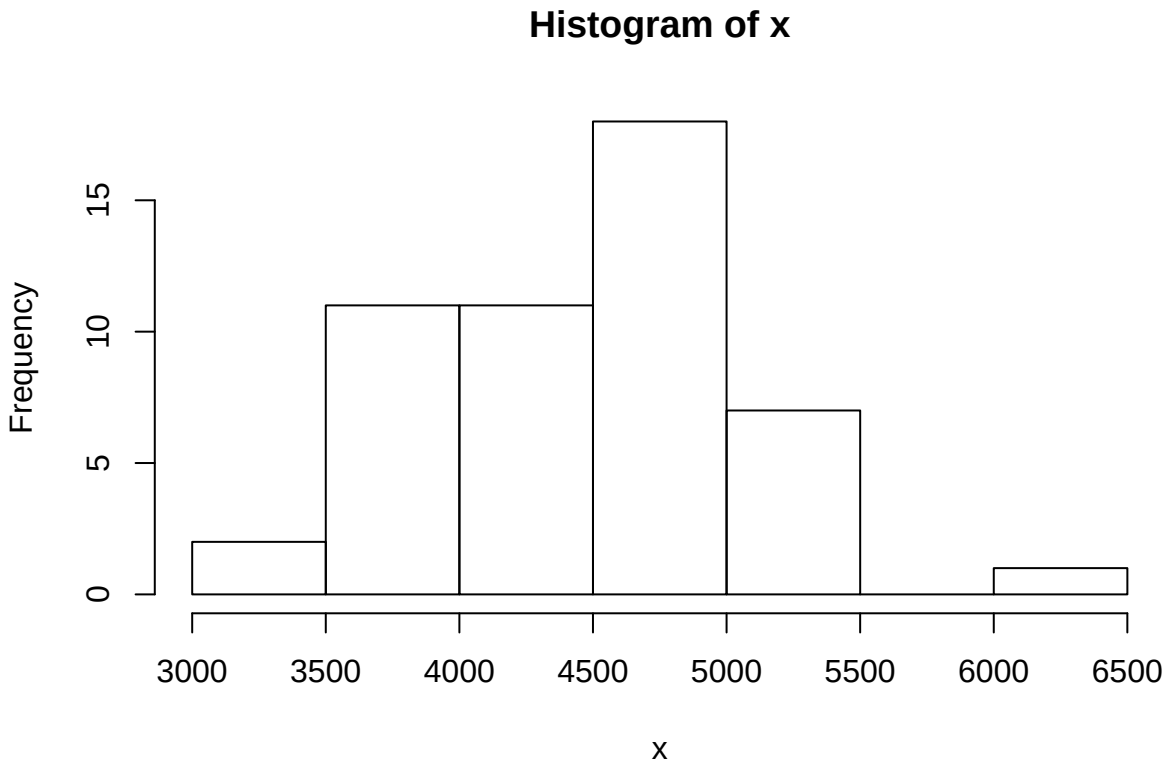
- High-level graphics
- Custom graphics
- Layered graphics in `ggplot2`

Functions for graphics

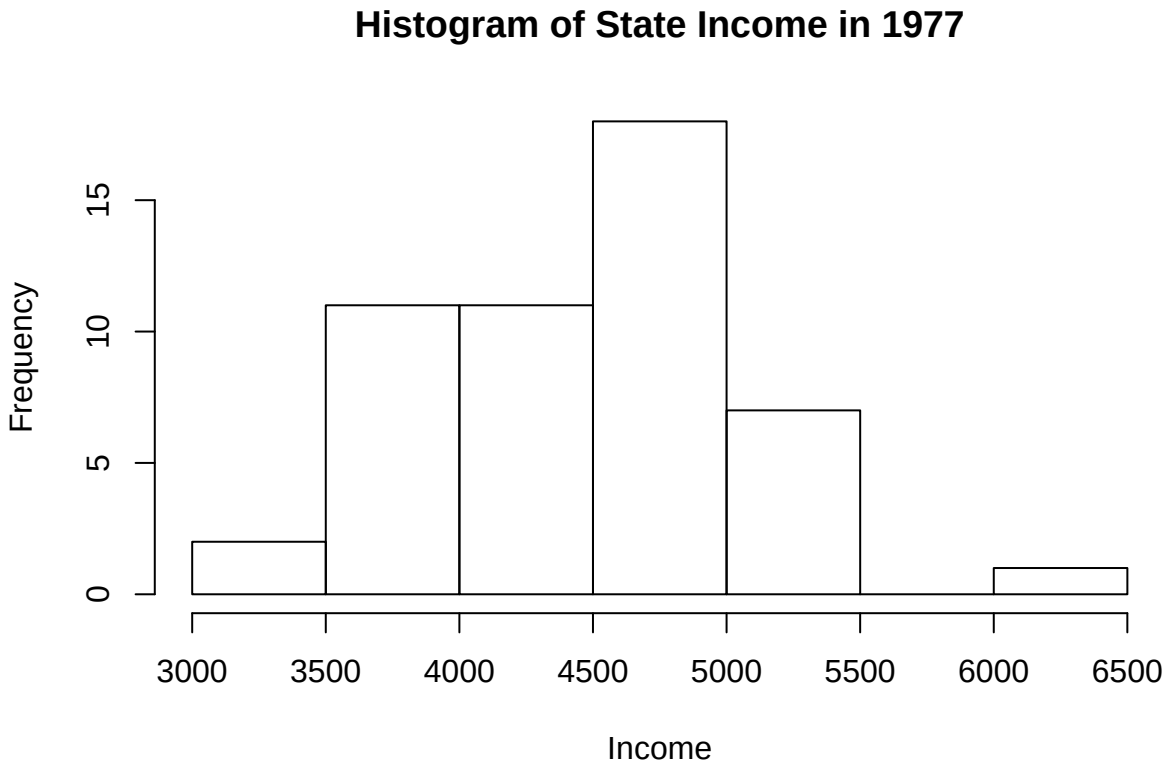
- The functions `hist()`, `boxplot()`, `plot()`, `points()`, `lines()`, `text()`, `mtext()`, `axis()`, etc. form a suite that plot graphs and add features to the graph
- Each of these functions have various options, to learn more about them, use the help
- `par()` can be used to set or query graphical parameters

Univariate data: Histogram

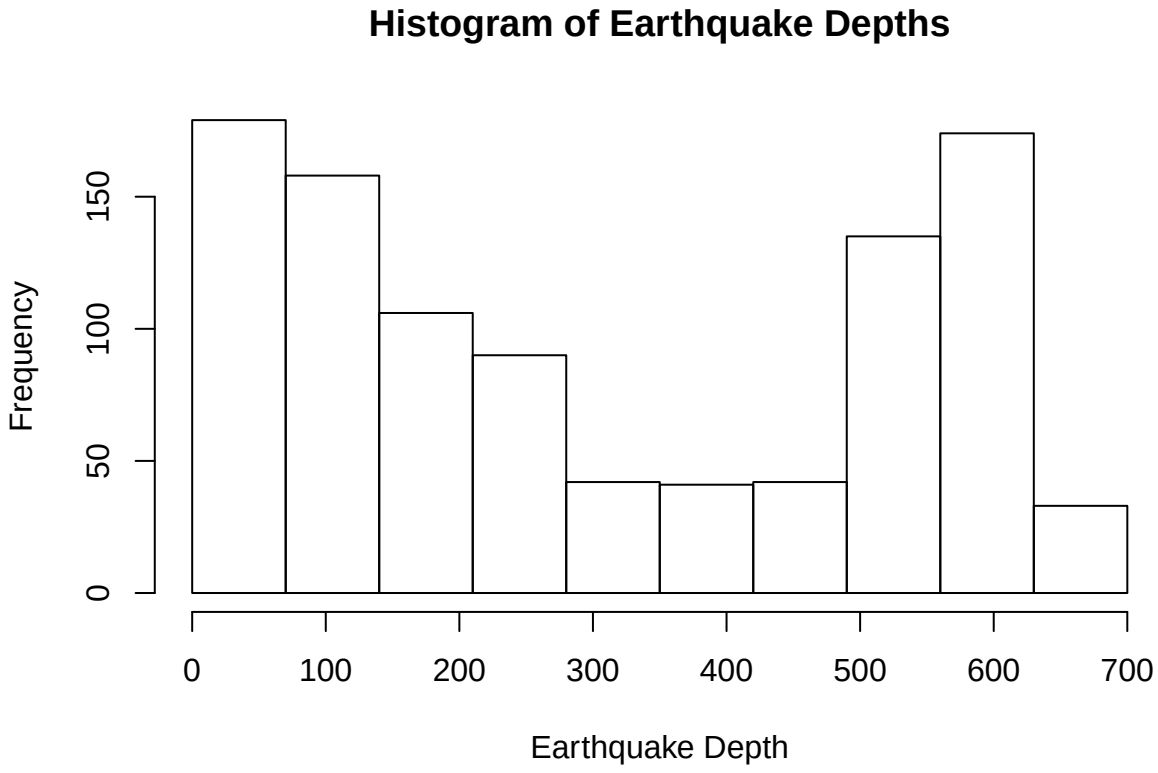
```
x = state.x77[, 2]  # 50 average state incomes in 1977
hist(x)
```




```
hist(x, breaks = 8, xlab = "Income", main = "Histogram of State Income in 1977")
```



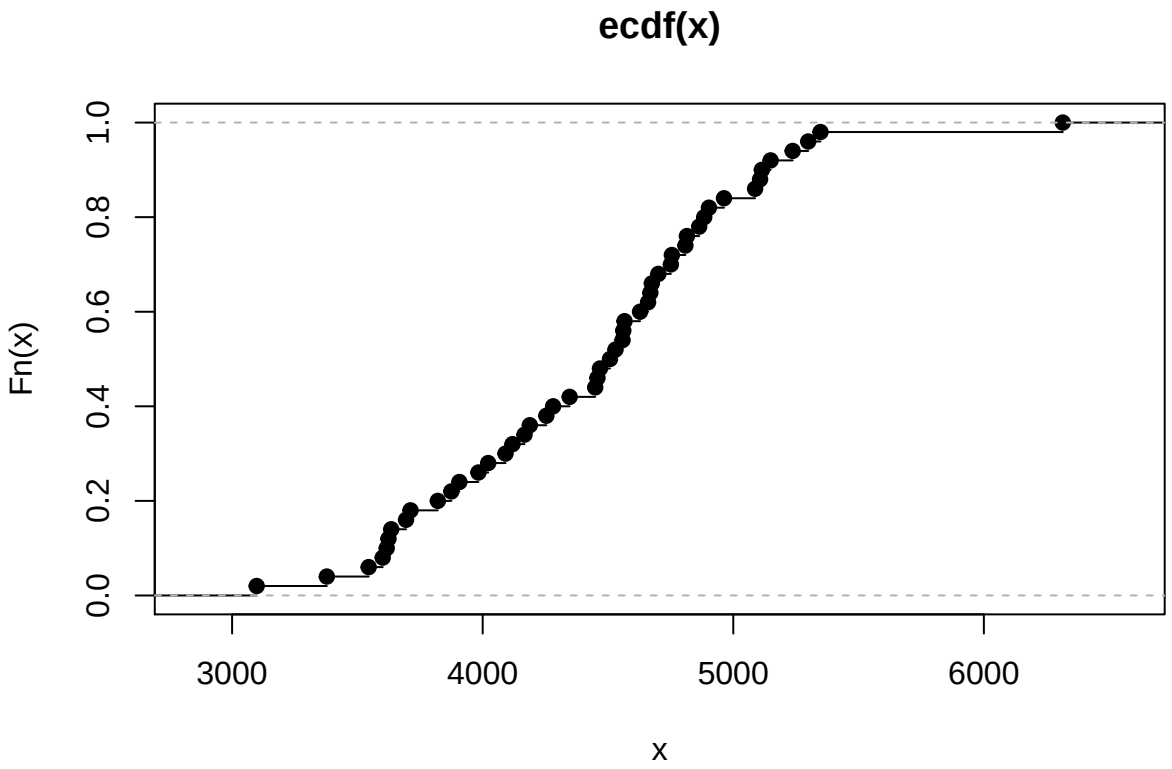

```r
y = quakes$depth  # 1000 earthquake depths
hist(y, seq(0, 700, by = 70), xlab = "Earthquake Depth", main = "Histogram of Earthquake Depths")
```



Empirical CDF

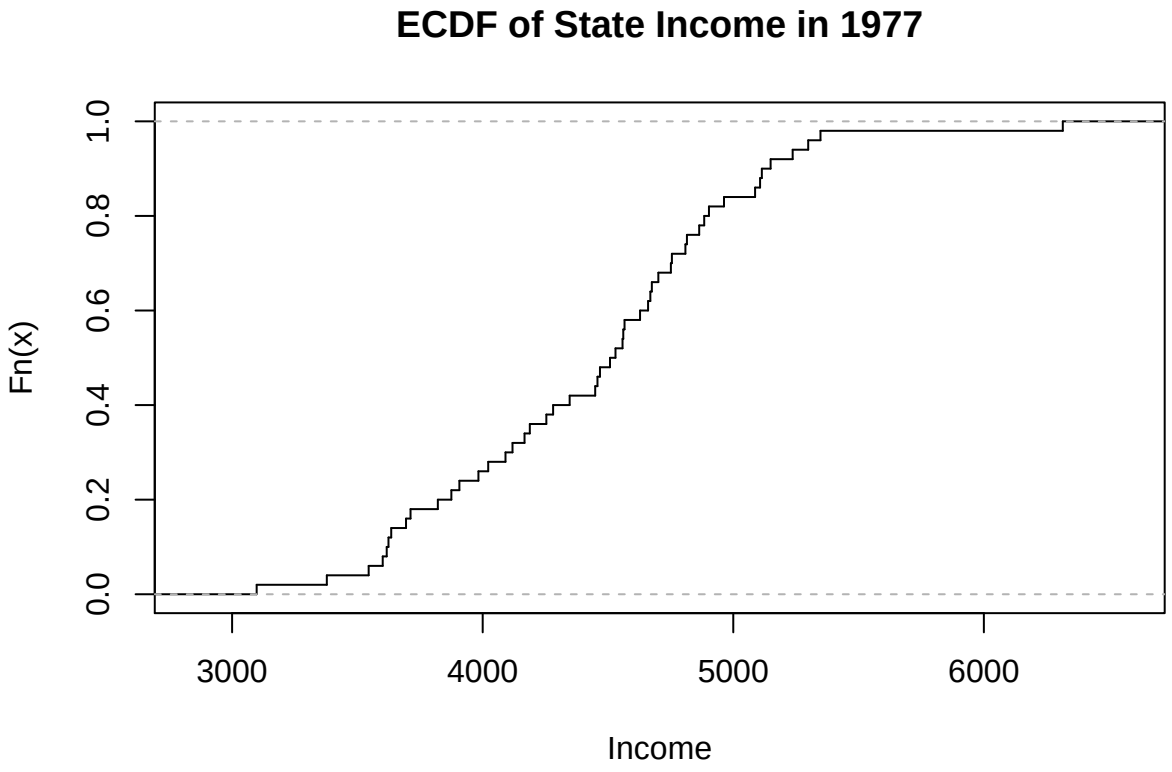
Function `ecdf()` provides data for empirical cdf

```
plot.ecdf(x)
```

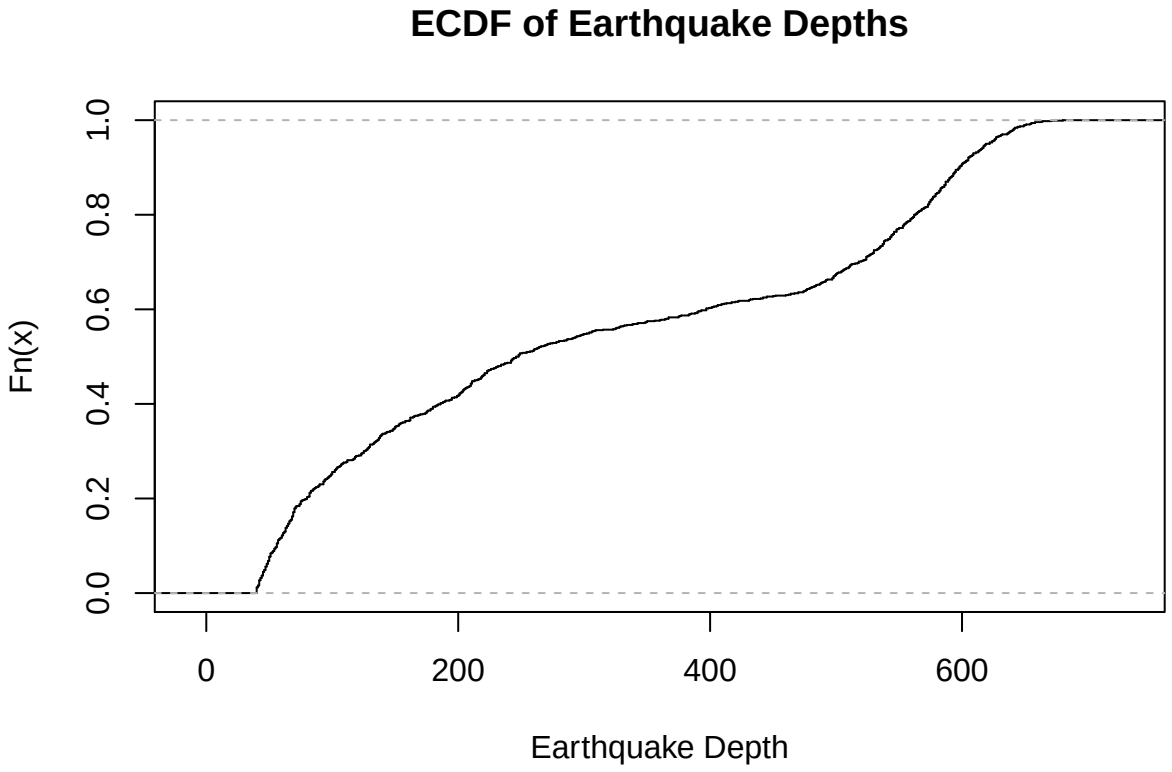


Can add vertical lines and remove dots

```
plot.ecdf(x, verticals = T, pch = "", xlab = "Income", main = "ECDF of State Income in 1977")
```



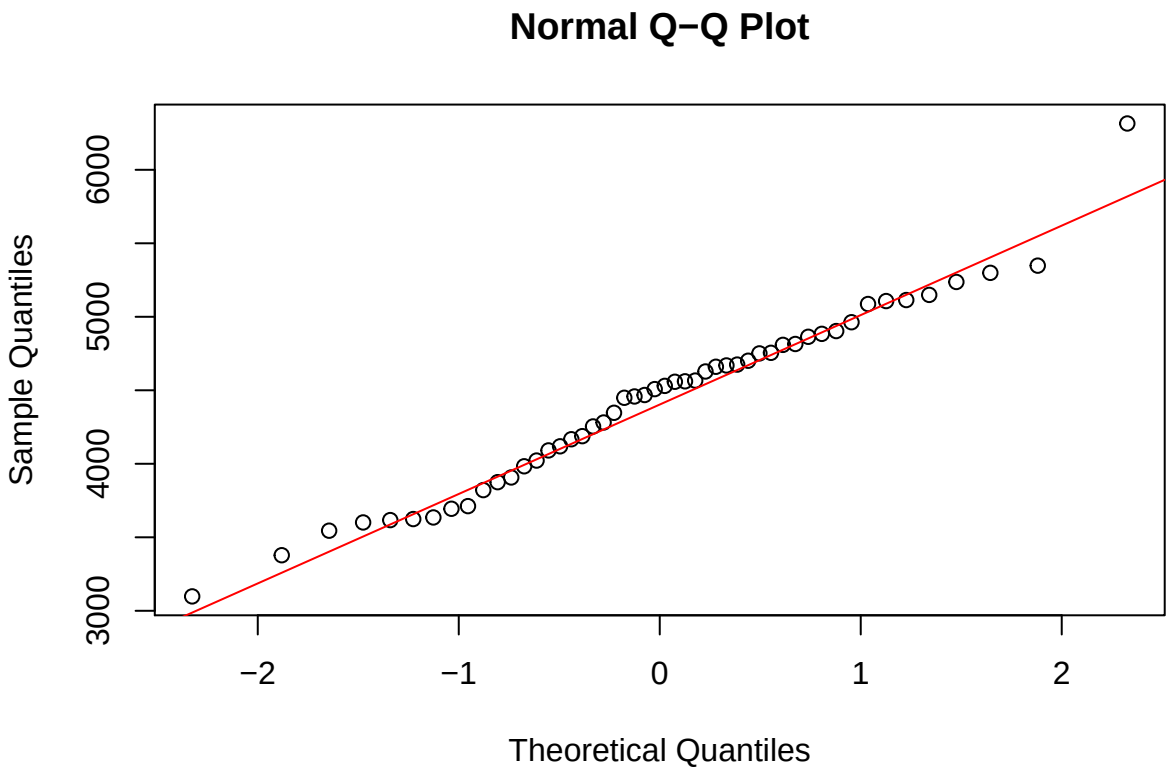

```
plot.ecdf(y, verticals = T, pch = "", xlab = "Earthquake Depth", main = "ECDF of Earthquake Depths")
```



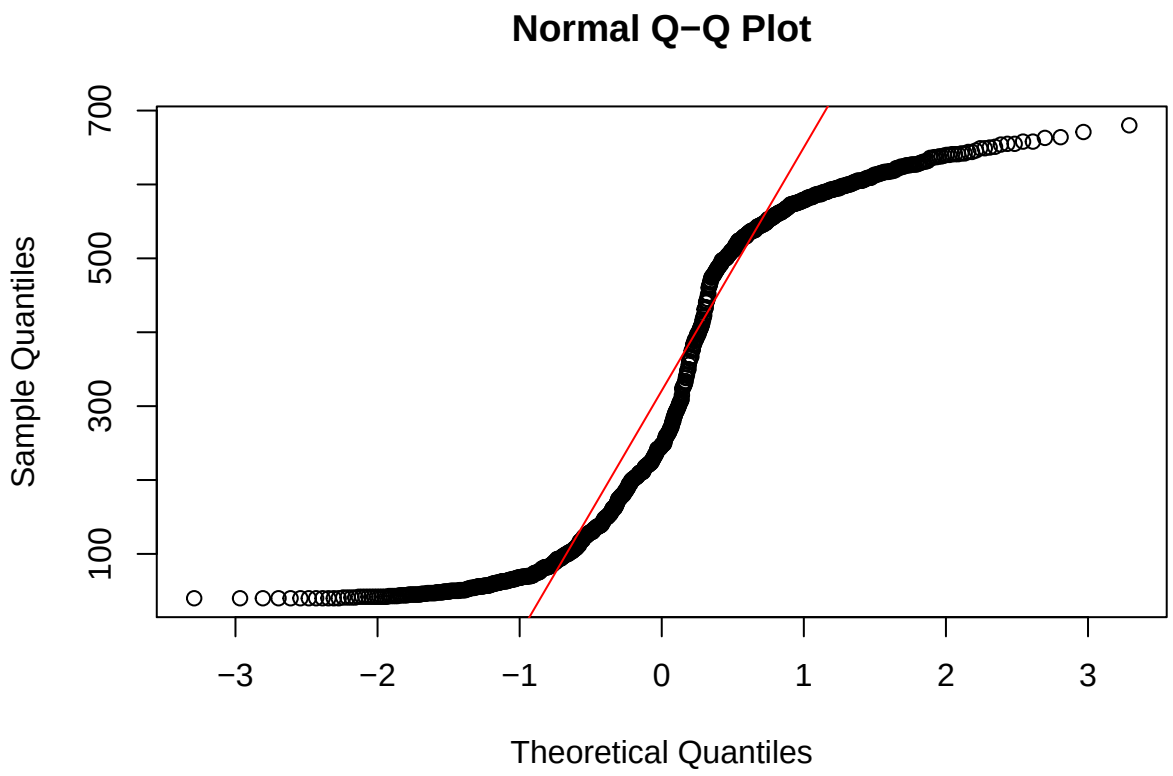
QQ Plots

- ‘qqnorm() plots the quantiles of a data set against the quantiles of a Normal distribution
- ‘qqplot() plots the quantiles of a first data set against the quantiles of a second data set

```
qqnorm(x)  # qq plot for the earthquake depths
qqline(x, col = "red")  # red reference line
```

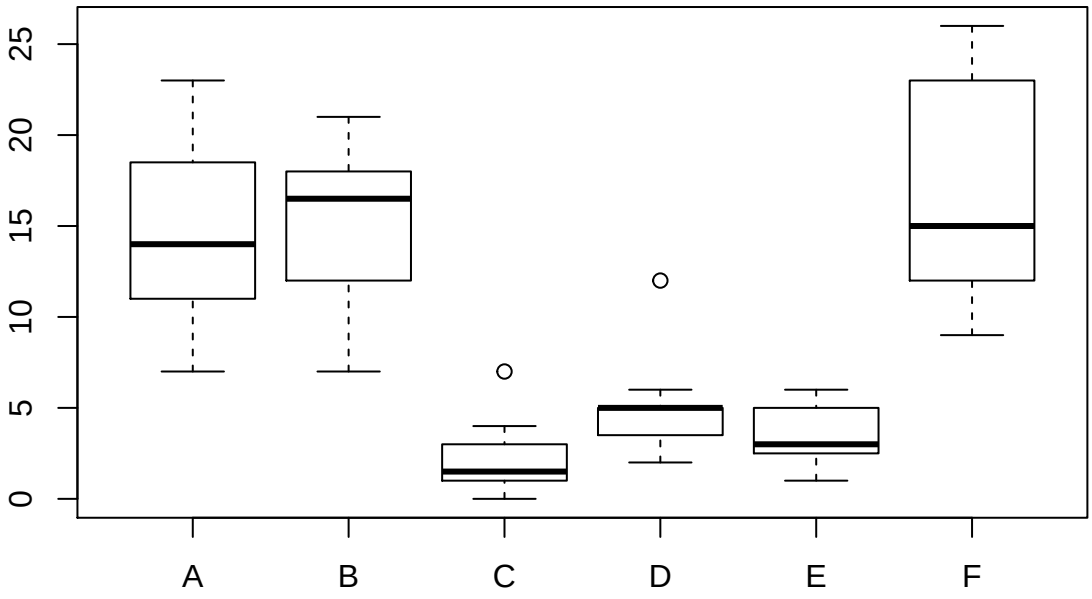



```
qqnorm(y)  # qq plot for the earthquake depths
qqline(y, col = "red")  # red reference line
```



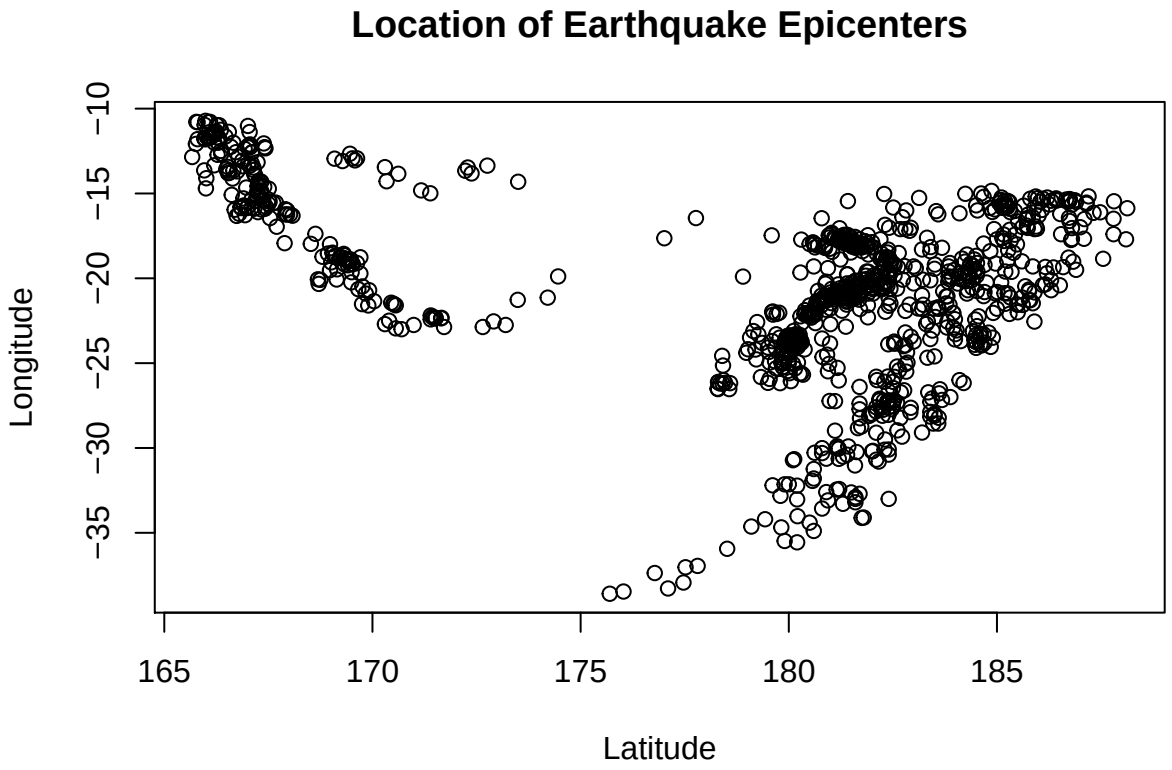
Box plots

```
boxplot(count ~ spray, data = InsectSprays)
```

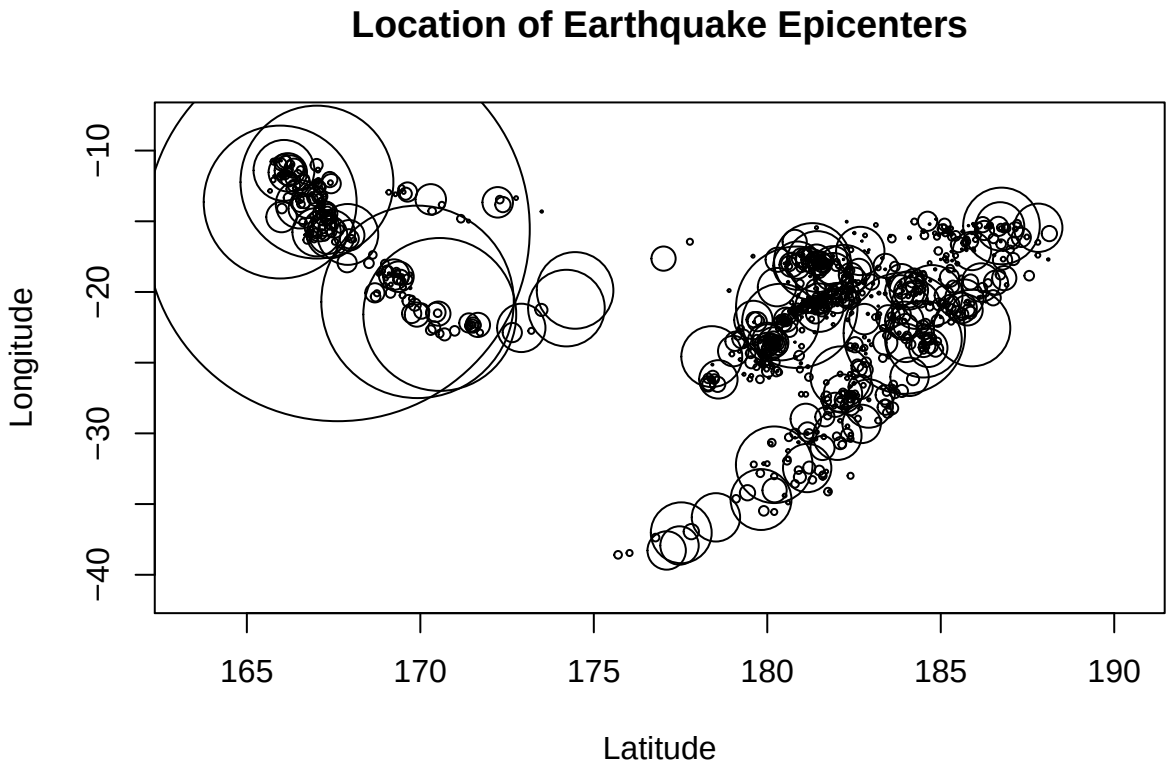


Scatterplots: `plot(x, y)`

```
plot(quakes$long, quakes$lat, xlab = "Latitude", ylab = "Longitude", main = "Location of Earthquake Epi
```



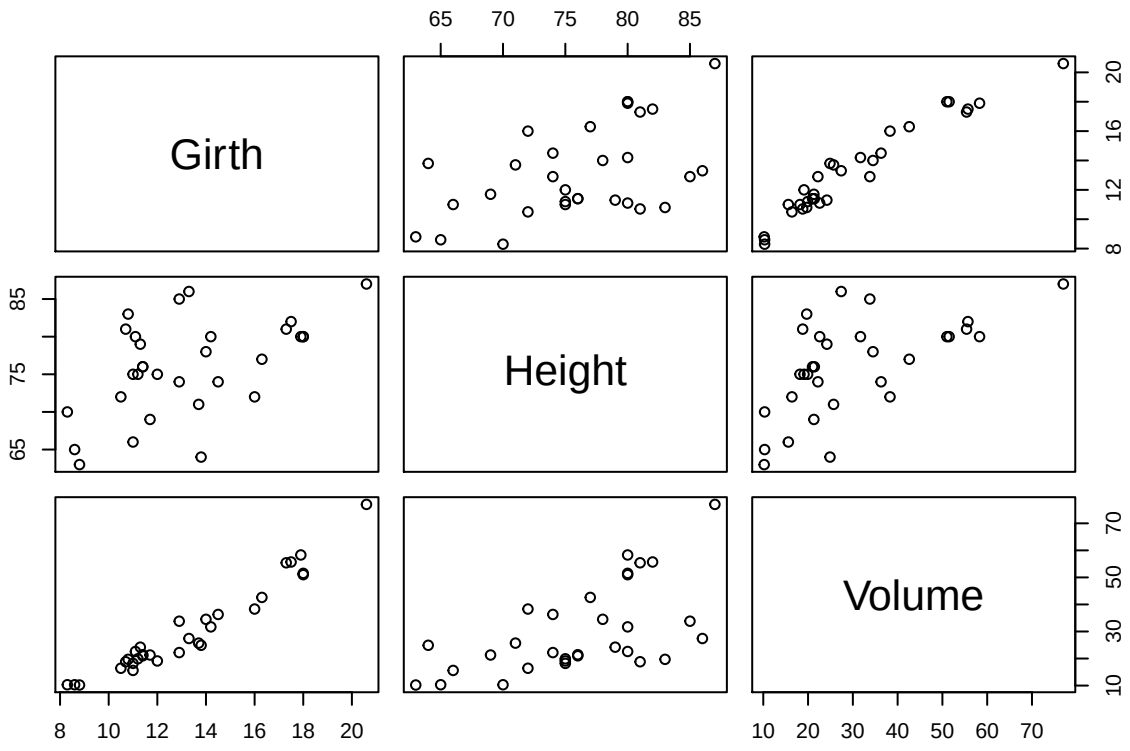

```
symbols(quakes$long, quakes$lat, circles = 10^quakes$mag, xlab = "Latitude",
    ylab = "Longitude", main = "Location of Earthquake Epicenters")
```



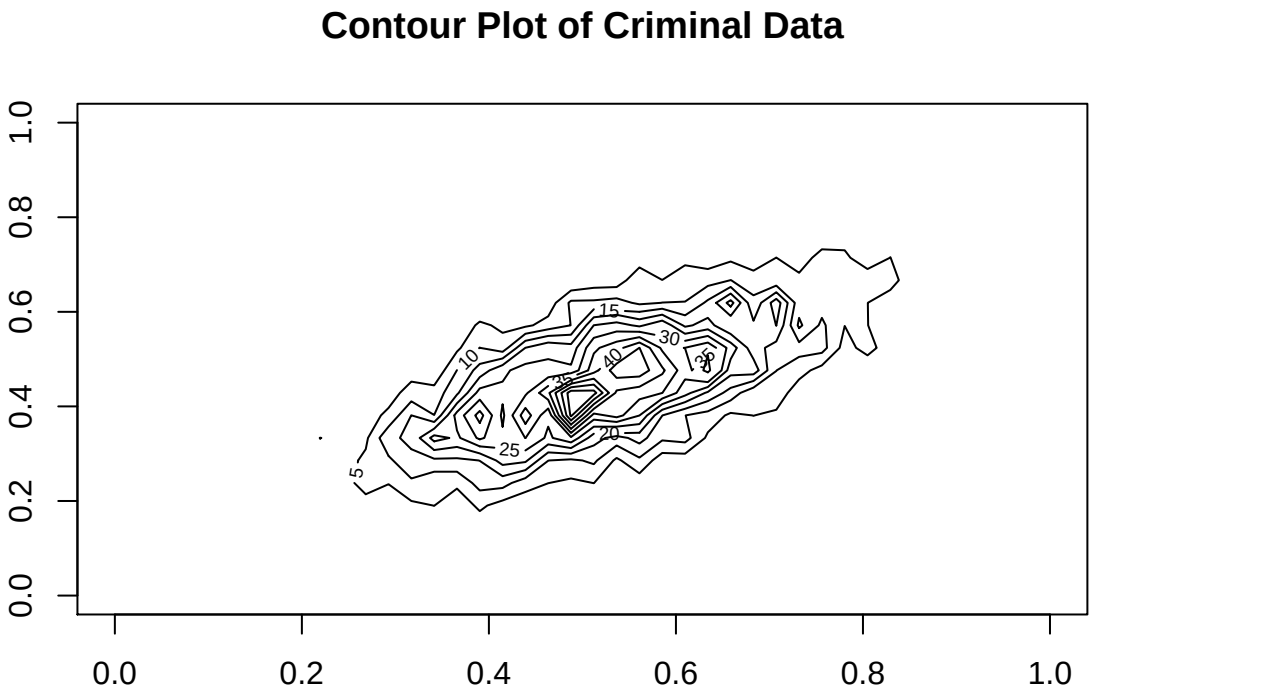
Three-dimensional data:

Showing `pairs(x)`

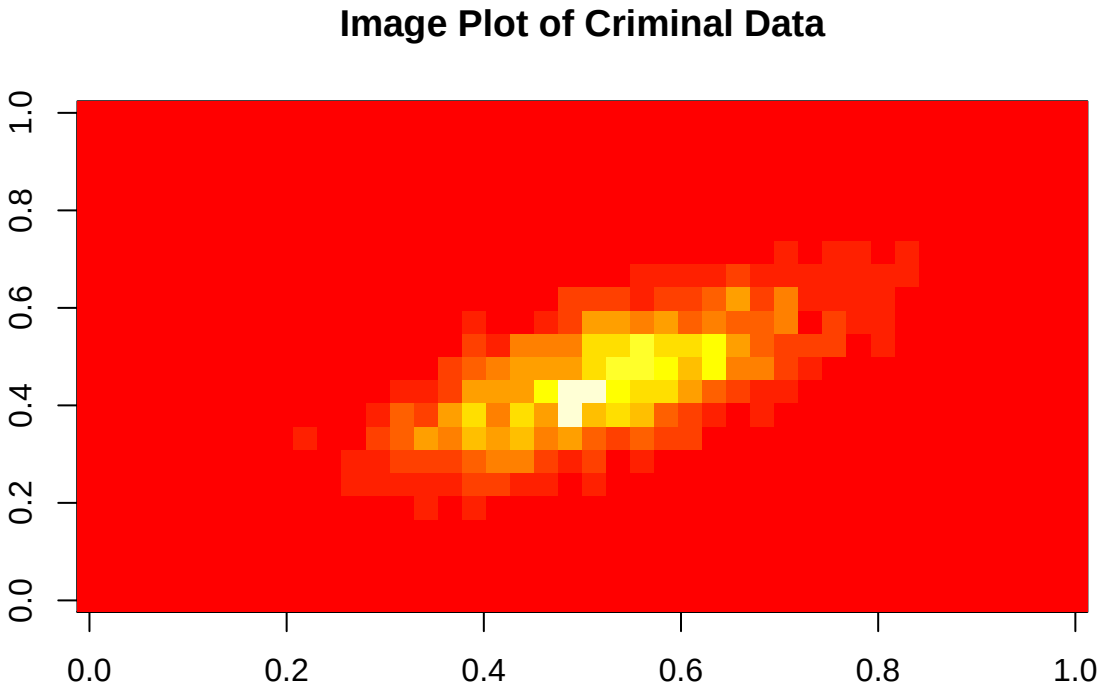
```
pairs(trees)
```



```r
contour(crimtab, main = "Contour Plot of Criminal Data")
```

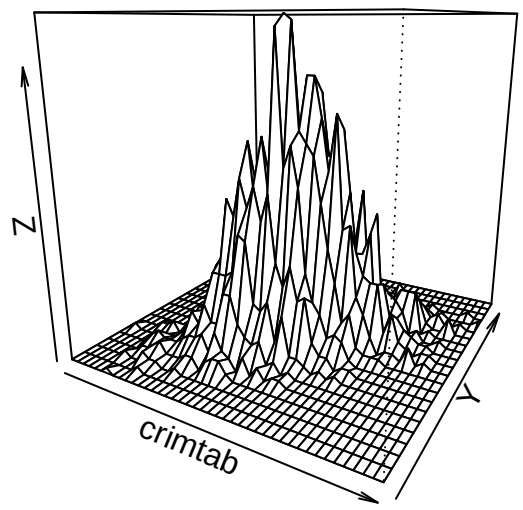



```r
image(crimtab, main = "Image Plot of Criminal Data")
```



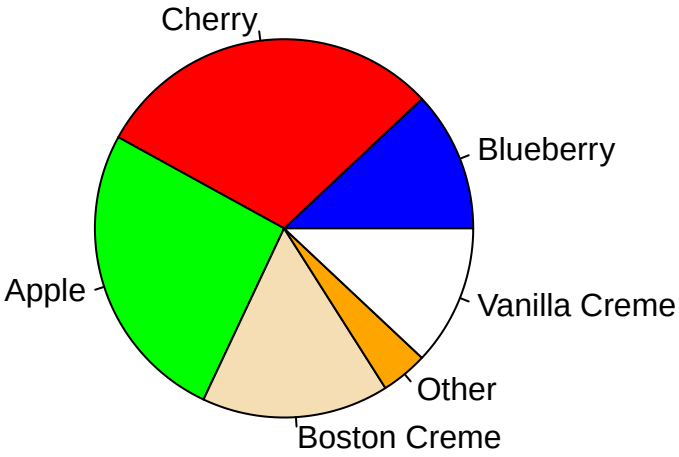

```r
persp(crimtab, theta = 30, main = "Perspective Plot of Criminal Data")
```

Perspective Plot of Criminal Data



Categorical data: Pie charts

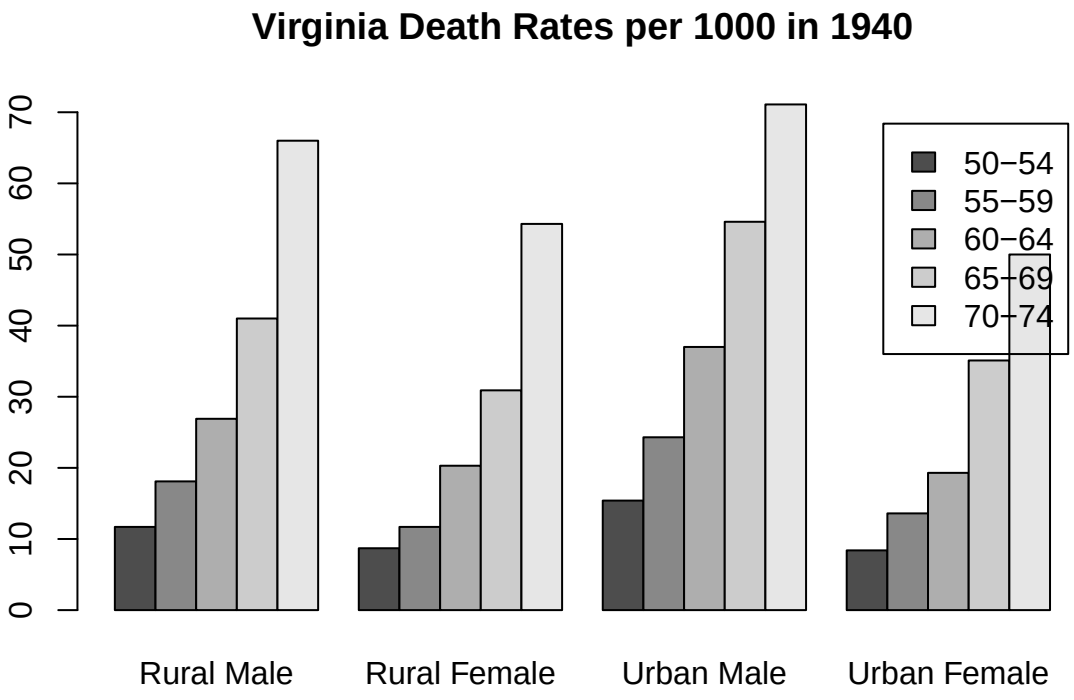
```
pie.sales = c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)
names(pie.sales) = c("Blueberry", "Cherry", "Apple", "Boston Creme", "Other",
    "Vanilla Creme")
pie(pie.sales, col = c("blue", "red", "green", "wheat", "orange", "white"))
```



Categorical data: Pie charts

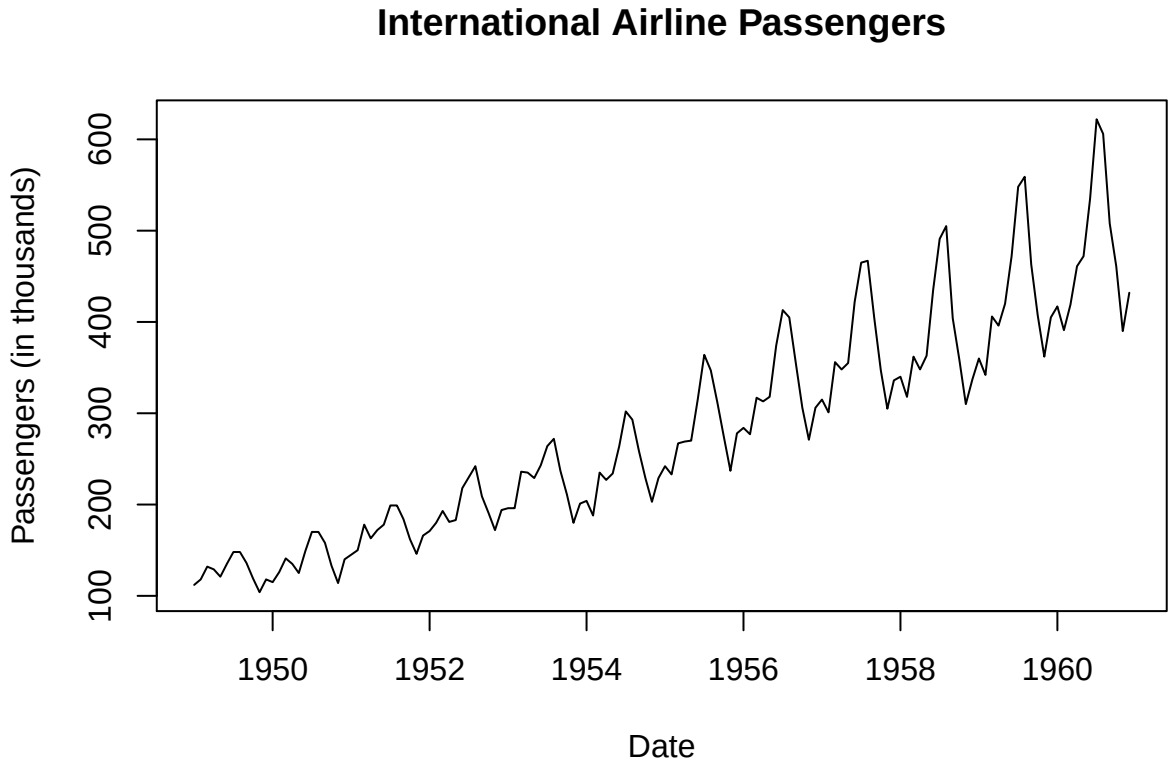
`dotchart()` and `barplot()` also available

```
barplot(VADeaths, beside = T, legend = T, main = "Virginia Death Rates per 1000 in 1940")
```

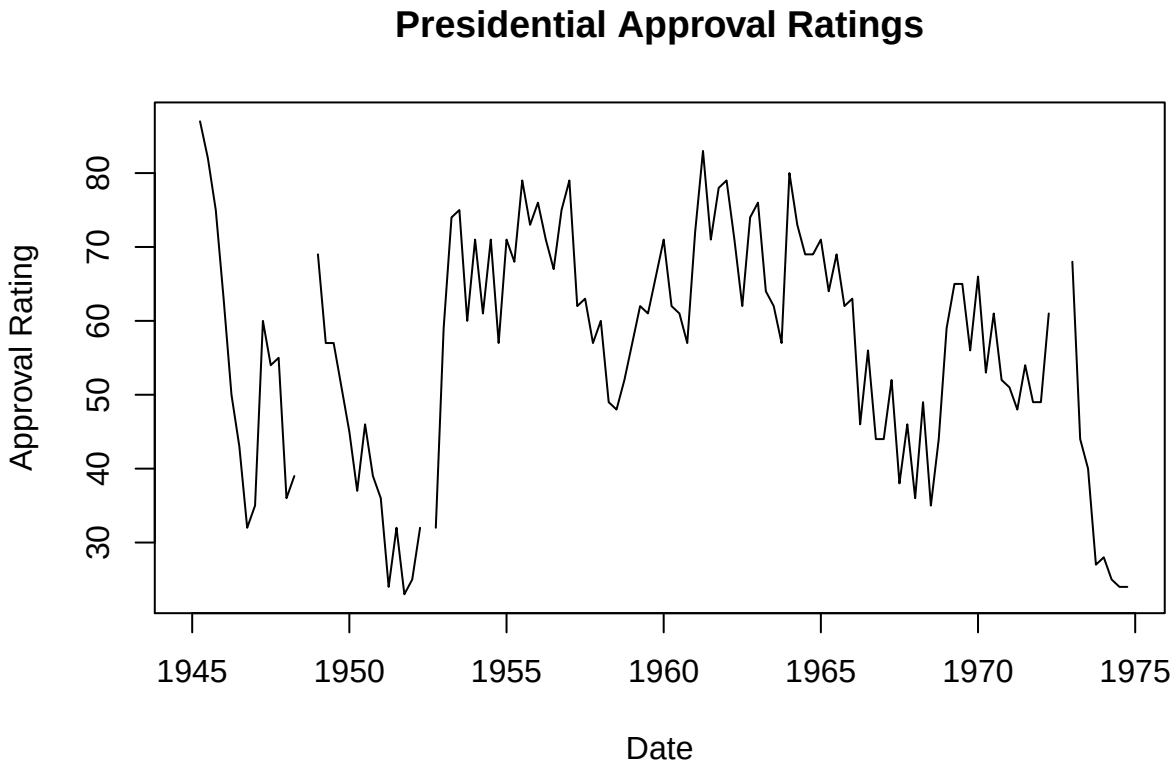


Time series plots

```
ts.plot(AirPassengers, xlab = "Date", ylab = "Passengers (in thousands)", main = "International Airline
```




```
ts.plot(presidents, xlab = "Date", ylab = "Approval Rating", main = "Presidential Approval Ratings")
```



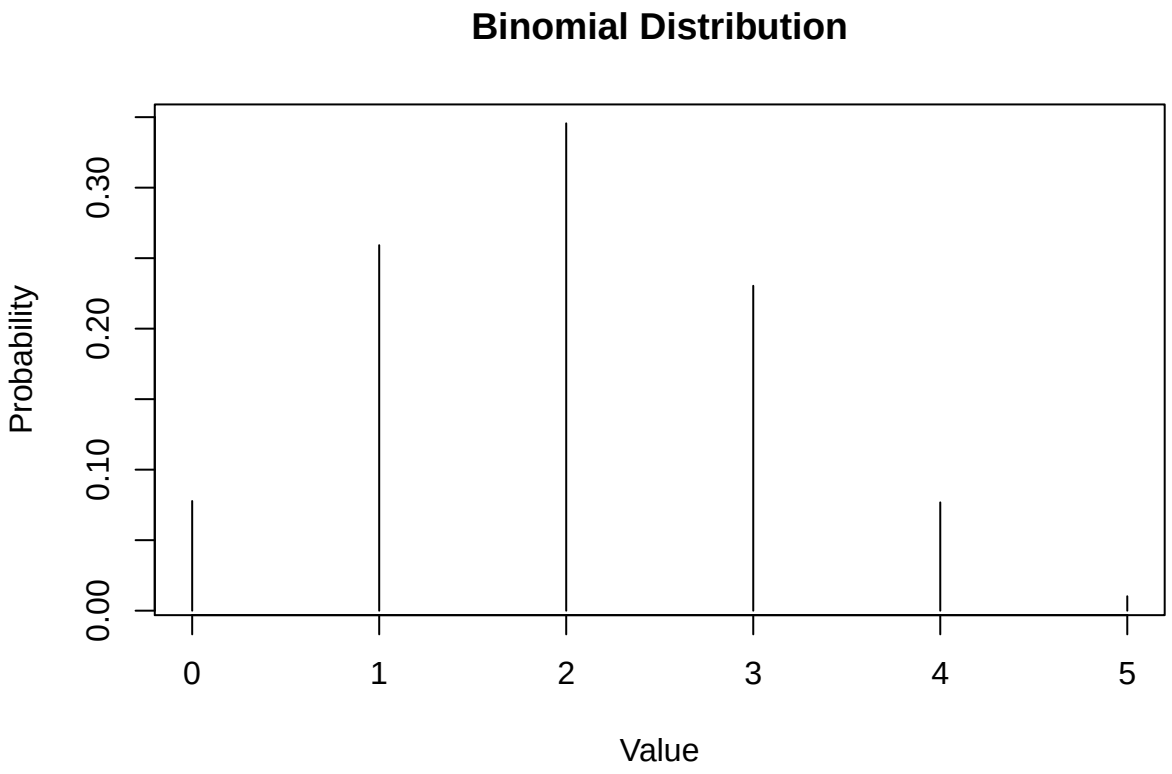
Custom graphics

- `par()` can be used to set or query graphical parameters
- We've already used some of these
- `adj`: text justification
- `bg`: background color
- `col`, `col.axis`, `col.lab`, ...: color specification
- `lty`: line type, e.g. dashed, dotted, solid (default), longdash, ...
- `lwd`: line width (helpful to increase for presentation plots)
- `mfcol` and `mfrow`: subsequent figures will be drawn in an nr-by-nc array on the device
- `pch`: point types
- `xlog`: plots to log scale if `TRUE`
- ...

Binomial distribution

Plot of binomial distribution with $n = 5$ and $p = .4$

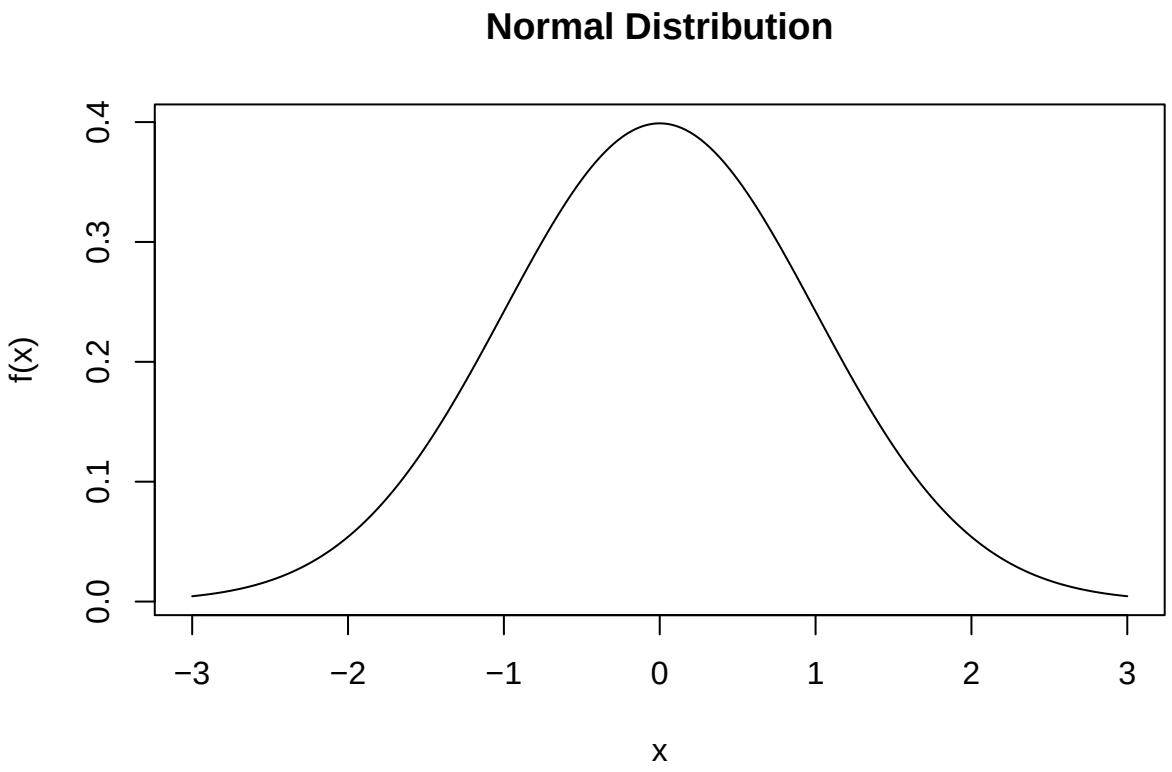
```
x = 0:5
y = dbinom(x, 5, 2/5)
plot(x, y, type = "h", main = "Binomial Distribution", xlab = "Value", ylab = "Probability")
```



Normal distribution

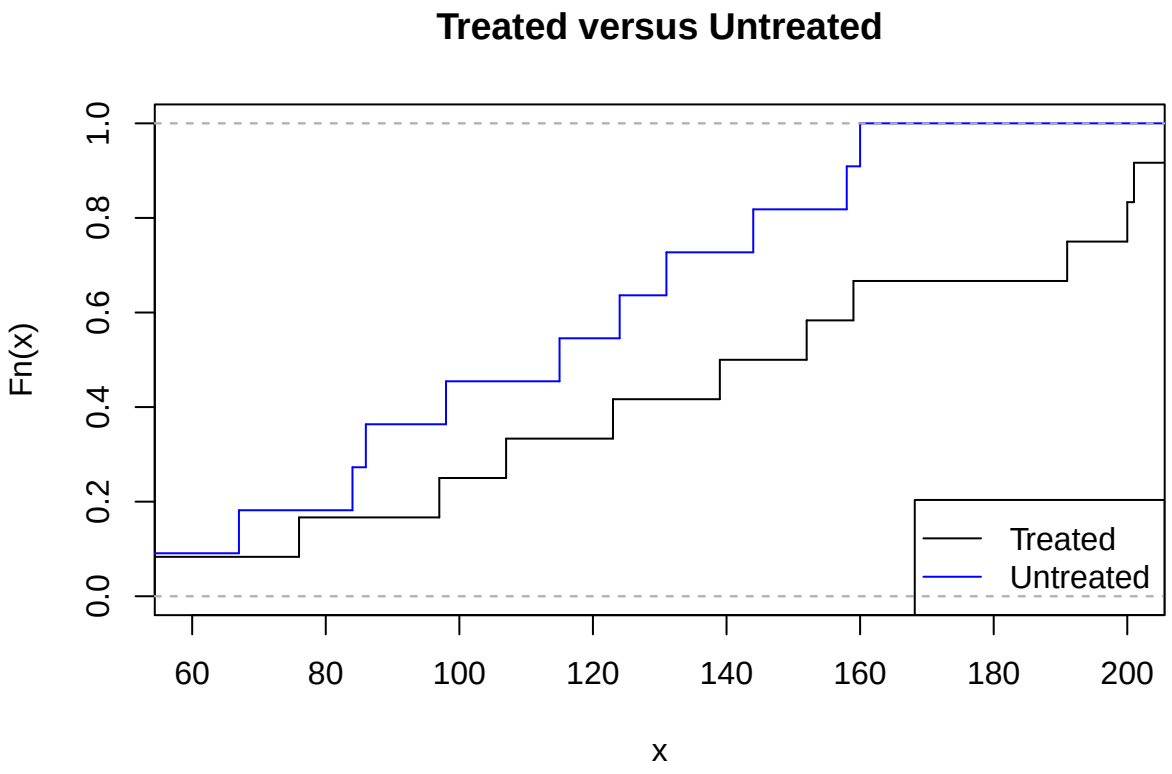
Probability density function for the standard Normal distribution from -3 to 3

```
x = seq(-3, 3, by = 0.01)
y = dnorm(x)
plot(x, y, type = "l", main = "Normal Distribution", ylab = "f(x)")
```



Two empirical cdfs: `Puromycin` dataset

```
x = Puromycin$rate[Puromycin$state == "treated"]
y = Puromycin$rate[Puromycin$state == "untreated"]

plot.ecdf(x, verticals = TRUE, pch = "", xlim = c(60, 200), main = "Treated versus Untreated")
lines(ecdf(y), verticals = TRUE, pch = "", xlim = c(60, 200), col = "blue")
legend("bottomright", c("Treated", "Untreated"), pch = "", col = c("black",
    "blue"), lwd = 1)
```



Saving a plot to a file

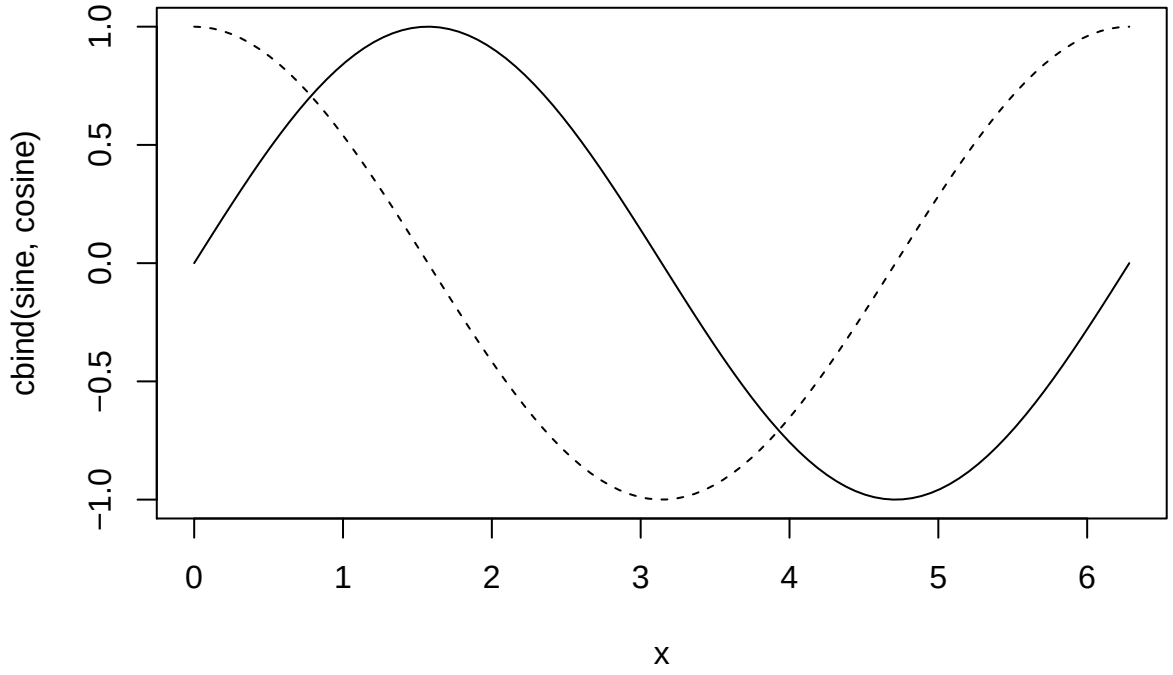
- Begin with functions `postscript()`, `pdf()`, `tiff()`,jpeg()', ...
- ... put all your plotting commands here ...
- Finish with `dev.off()`

```
pdf("2cdfs.pdf", width = 6, height = 4)
plot.ecdf(x, verticals = TRUE, pch = "", xlim = c(60, 200), main = "Treated versus Untreated")
lines(ecdf(y), verticals = TRUE, pch = "", xlim = c(60, 200), col = "blue")
legend("bottomright", c("Treated", "Untreated"), pch = "", col = c("black",
    "blue"), lwd = 1)
dev.off()
```

```
## pdf
##   2
```

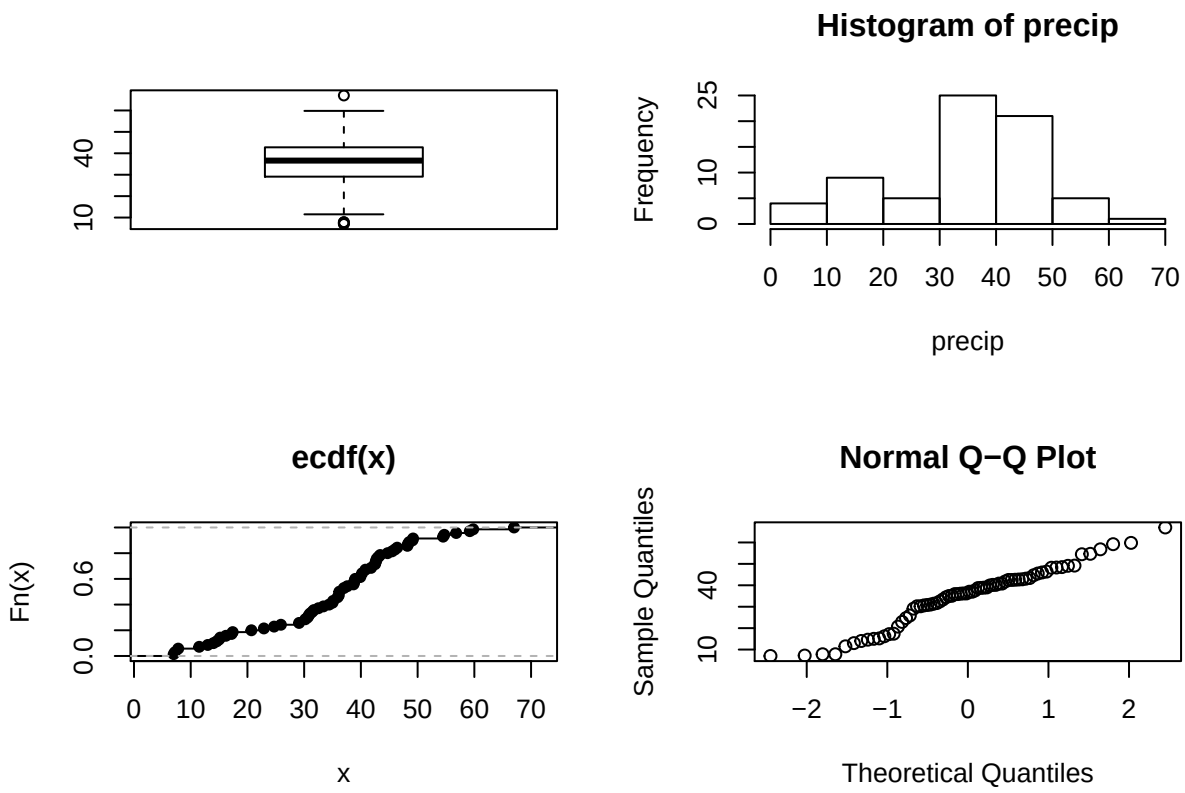
Multiple plots on one set of axes

```
x = seq(0, 2 * pi, length = 100)
sine = sin(x)
cosine = cos(x)
matplot(x, cbind(sine, cosine), col = c(1, 1), type = "l")
```



Multiple frame plots

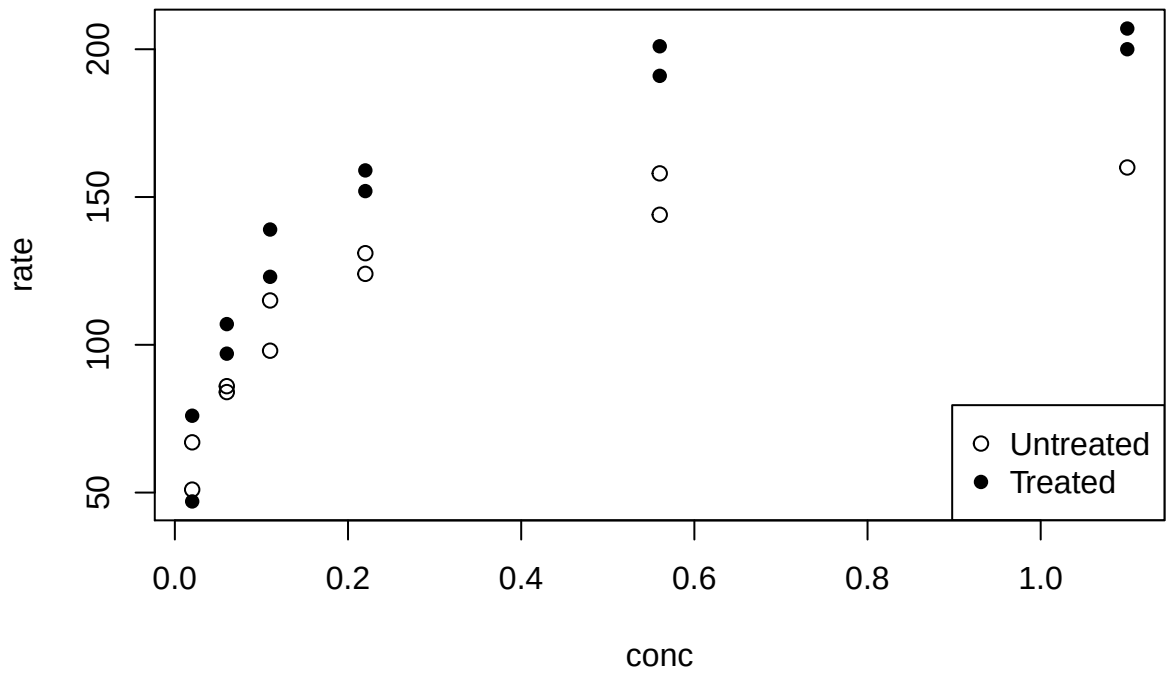
```
par(mfrow = c(2, 2))
boxplot(precip)
hist(precip)
plot.ecdf(precip)
qqnorm(precip)
```




```
par(mfrow = c(1, 1))
```

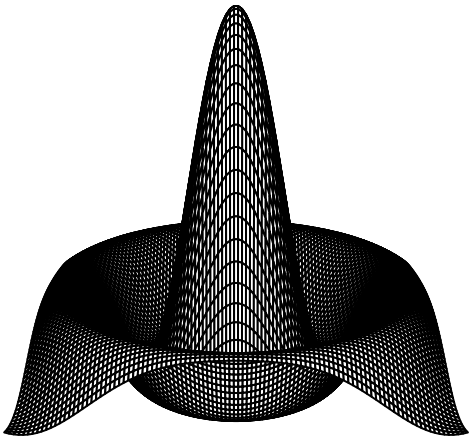
Plot using statistical model

```
plot(rate ~ conc, data = Puromycin, pch = 15 * (state == "treated") + 1)
legend("bottomright", legend = c("Untreated", "Treated"), pch = c(1, 16))
```

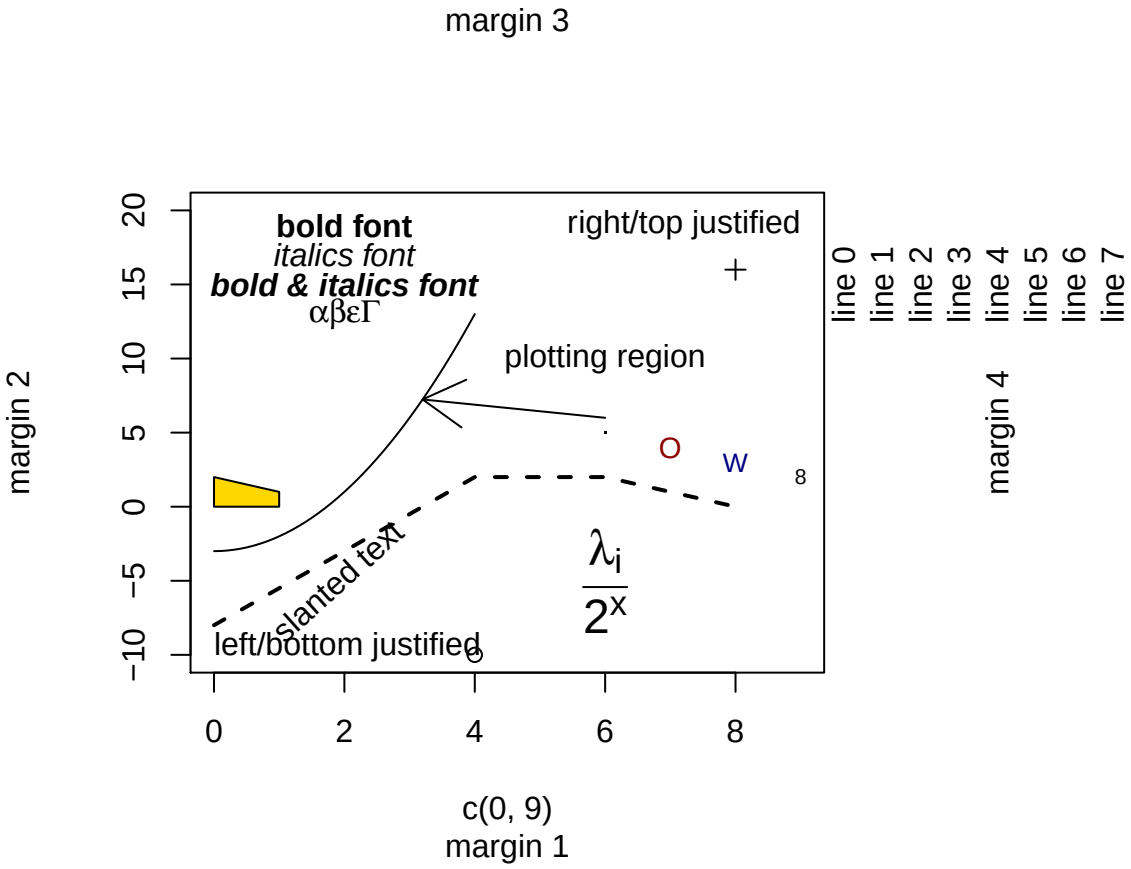


Plot using `persp()` for wire mesh

```
x = seq(-8, 8, length = 100)
y = x
f = function(x, y) sin(sqrt(x^2 + y^2))/(sqrt(x^2 + y^2))
z = outer(x, y, f)
persp(x, y, z, xlab = "", ylab = "", zlab = "", axes = F, box = F)
```



Custom plot: Leemis Chapter 21



ggplot2

- Everything so far has been part of base R
- The ggplot2 package is a popular by Hadley Wickham
- Based on the grammar of graphics, which tries to take the good parts of base and lattice graphics and none of the bad parts
- http://ggplot2.org

Forced Expiratory Volume (FEV) Data

Explore a data on the relationship between smoking and pulmonary function from Rosner (1999) using layered graphics created with ggplot2. The data consists of a sample of 654 youths, aged 3 to 19, in the area of East Boston during middle to late 1970's. Our main interest is in the relationship between smoking and FEV. Use the following commands to load the data and ggplot2.

```
load(url("http://www.faculty.ucr.edu/~jflegal/fev.RData"))
library(ggplot2)
str(fevdata)
```

```
## 'data.frame':    654 obs. of  5 variables:
##  $ age   : int  9 8 7 9 9 8 6 6 8 9 ...
##  $ fev   : num  1.71 1.72 1.72 1.56 1.9 ...
##  $ height: num  57 67.5 54.5 53 57 61 58 56 58.5 60 ...
##  $ sex   : Factor w/ 2 levels "female","male": 1 1 1 2 2 1 1 1 1 1 ...
##  $ smoke : Factor w/ 2 levels "nonsmoker","smoker": 1 1 1 1 1 1 1 1 1 1 ...
```

Layered graphics in ggplot2

ggplot2 allows you to construct multi-layered graphics. A plot in ggplot2 consists of several components:

- Defaults
- Layers
- Scales
- Coordinate system

Layers consist of:

- Data
- Mapping
- Geom
- Stat
- Position

ggplot2 uses the + operator to build up a plot from these components. The basic plot definition looks like this:

```
ggplot(data, mapping) + layer(stat = "", geom = "", position = "", geom_parms = list(),
    stat_params = list(), )
```

We usually won't write out the full specification of layer, but use shortcuts like:

```
geom_point()
stat_summary()
```

Every geom has a default stat and every stat has a default geom.

Usually, data and mappings are the same for all layers and so they can be stored as defaults:

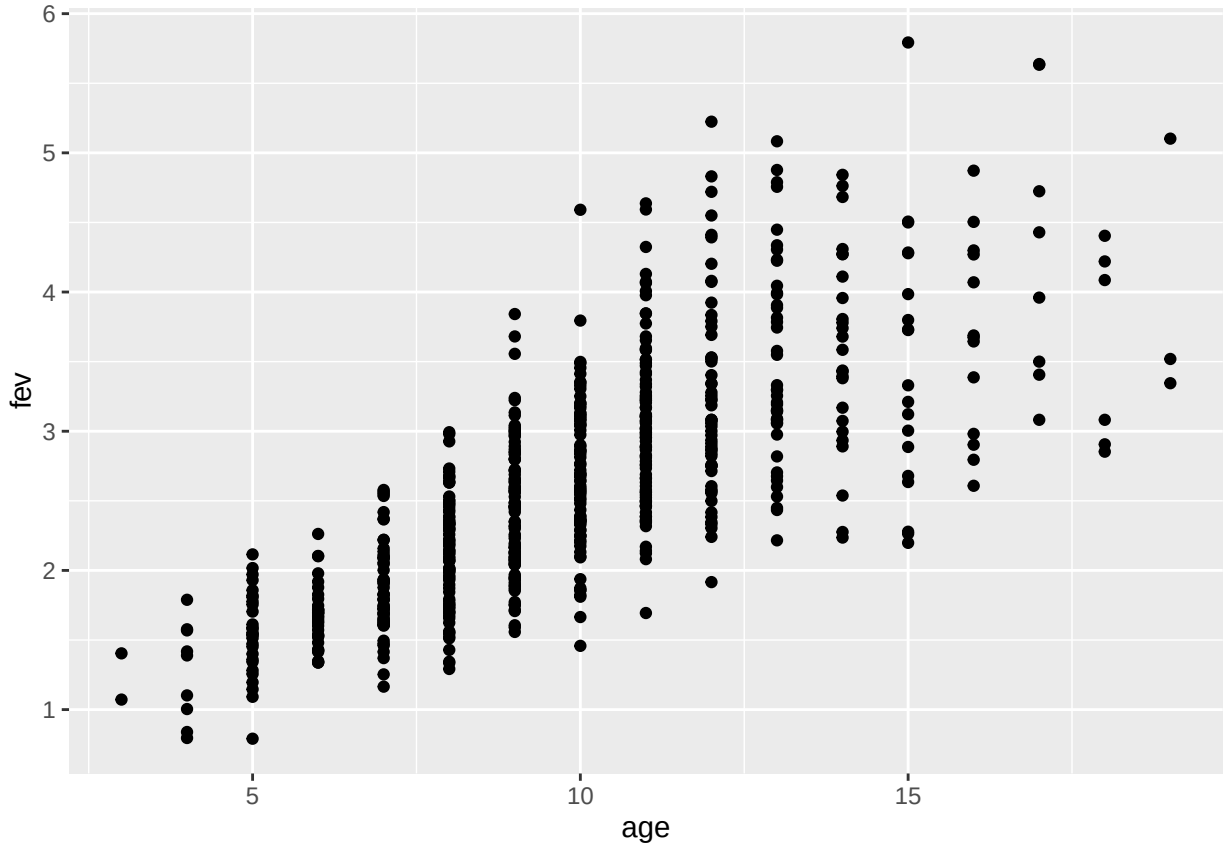
```
ggplot(data, mapping = aes(x = x, y = y))
```

All layers use the default values of data and mapping unless explicitly you override them explicitly. The `aes()` function describes the mapping that will be used for each layer. You must specify a default, but you can also specify per layer mappings and data:

```
ggplot(data, mapping(aes(x = x, y = y)))
+geom_point(aes(color = z))
+geom_line(data = another_data)
```

Scatterplot of `age` versus `fev`:

```
s <- ggplot(fevdata, aes(x = age, y = fev))
s + geom_point()
```



- What do you see?
- Try coloring the points according to sex or smoker at home. Are there any apparent differences?

Layering

You can add additional layers to a plot with the `+` operator. Let's try adding a line that shows the average value of `fev` for each `age`. One way to do this is to construct an additional data frame with columns corresponding to `age` and average value of `fev` and then add a layer with this data. We will do this with the `dplyr` package. Don't worry about how it works for now. We will learn more about it later. First, you will need to install `dplyr` if it is not already installed:

```
install.packages("dplyr")
```

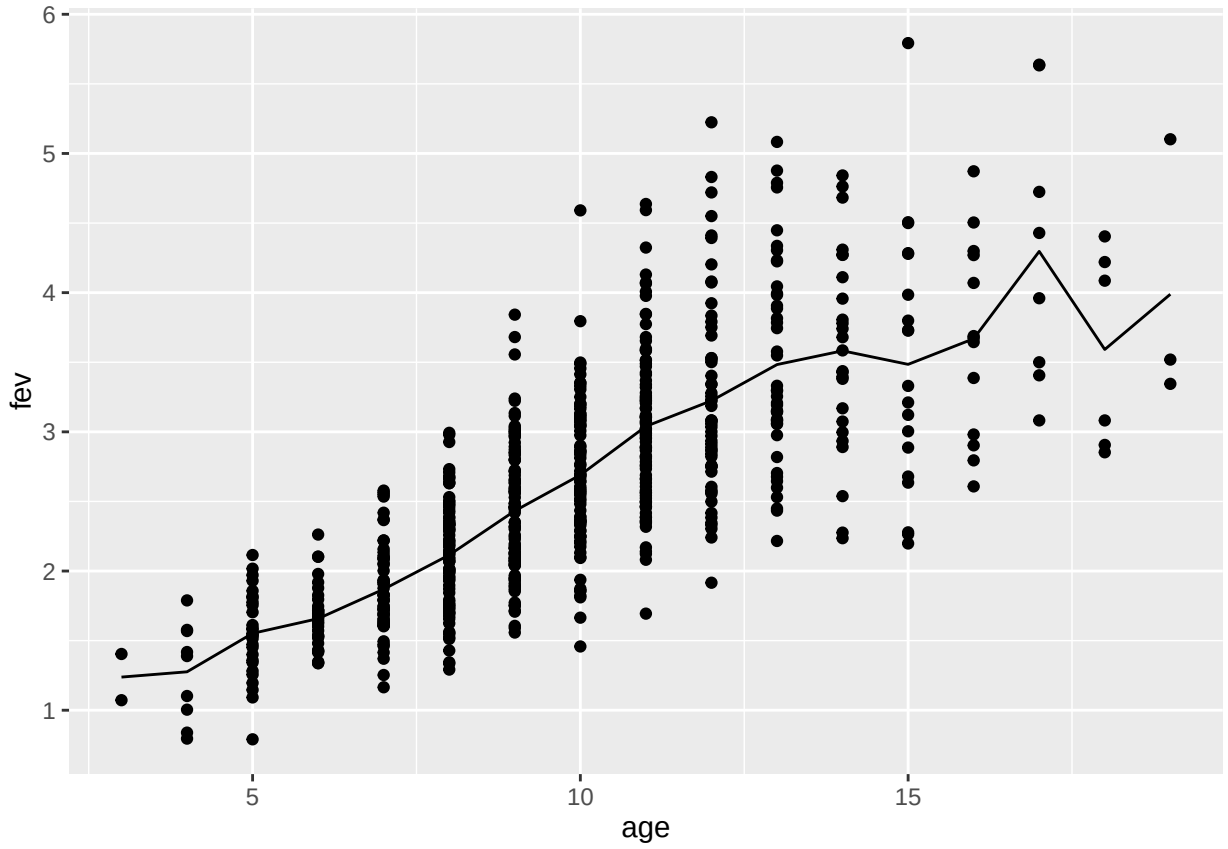
```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
fev_mean <- summarize(group_by(fevdata, age), fev = mean(fev))
s + geom_point() + geom_line(data = fev_mean)
```



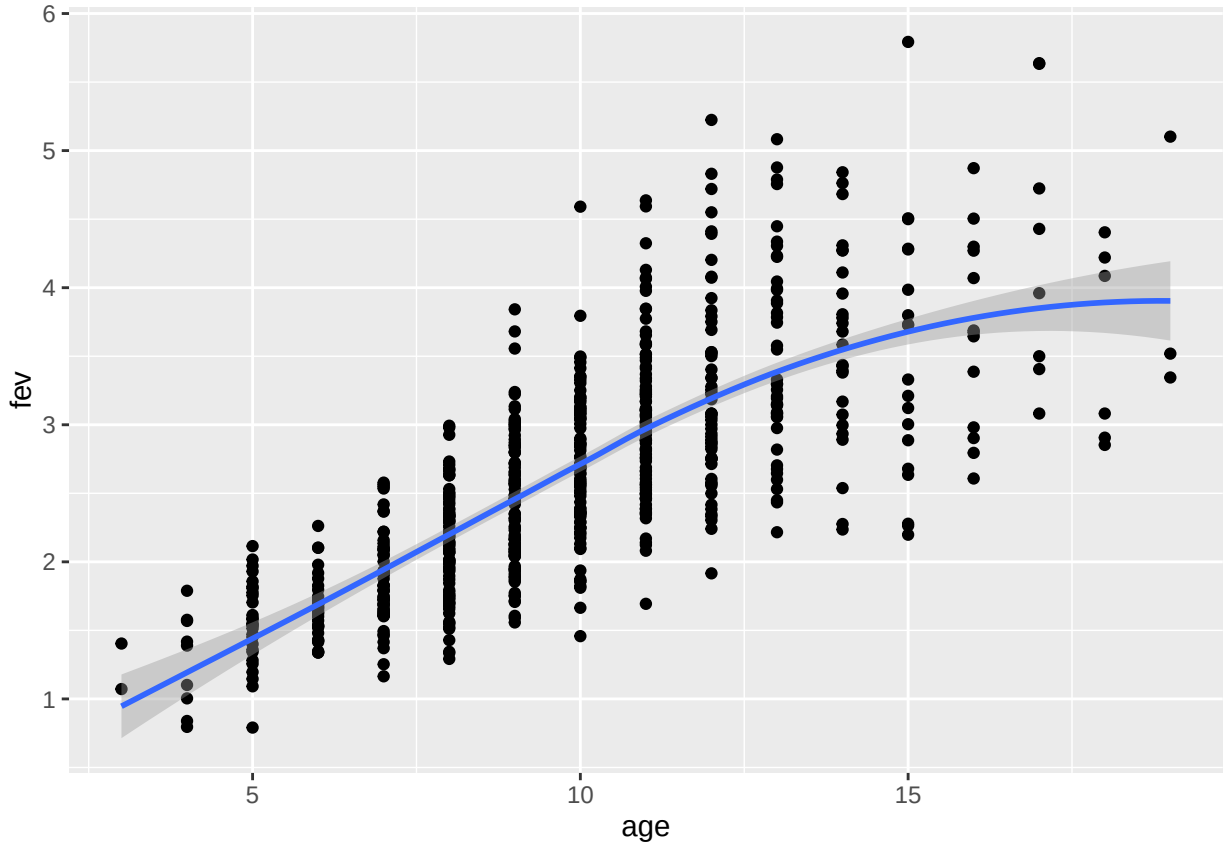
Smoothing

- Similarly, we can add a smoother to the scatterplot by first computing the smooth and storing it in data frame. Then add a layer with that data. Since smoothers are so useful, this operation is available in `ggplot2` as a stat.
- `stat_smooth()` provides a smoothing transformation. It creates a new data frame with the values of the smooth and by default uses `geom="ribbon"` so that both the smooth curve and error bands are shown.

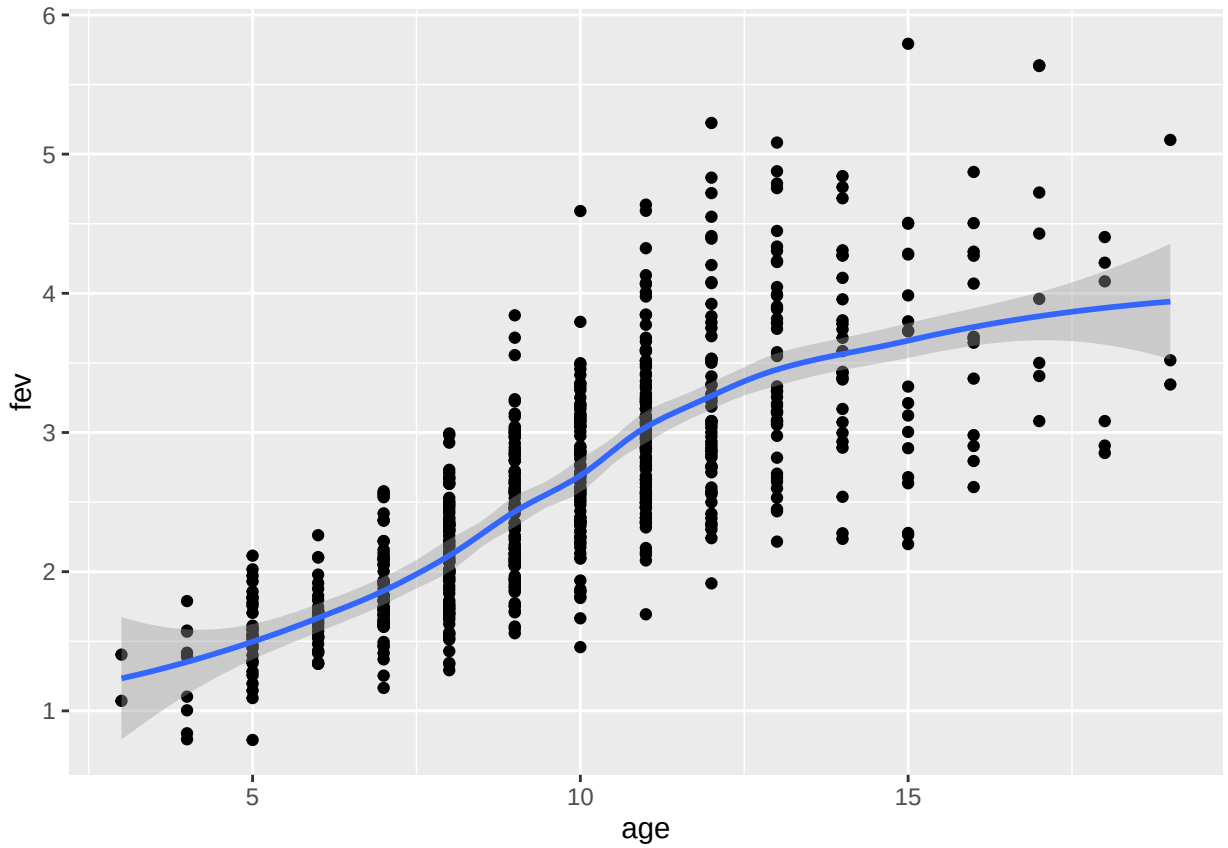
```
s + geom_point() + stat_smooth()
```

- The default smoother is loess. This is the name given to the locally weighted quadratic regression smoother with tricubic weight function. Its bandwidth can be specified indirectly with the span parameter.

```
s + geom_point() + stat_smooth(span = 1)
```

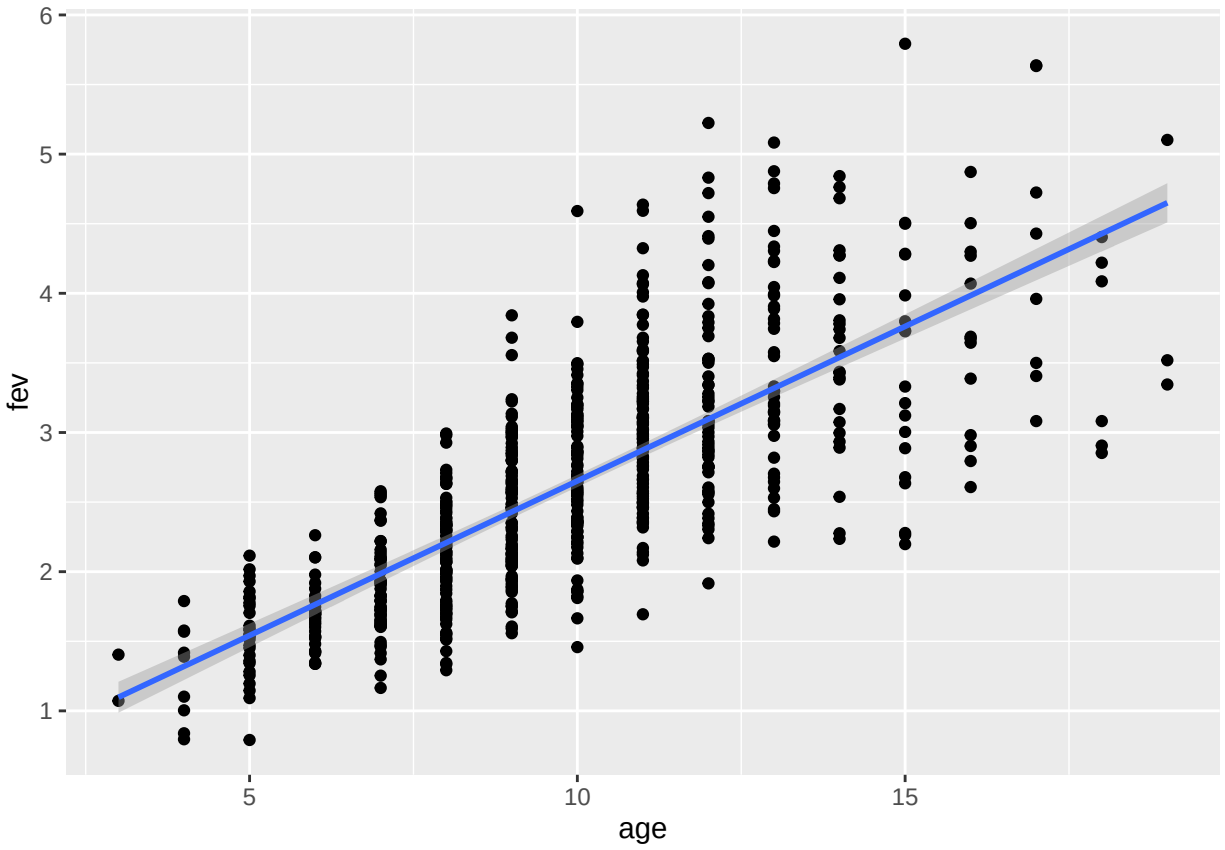



```
s + geom_point() + stat_smooth(span = 1/2)
```



- We could also use linear regression by specifying `lm`:

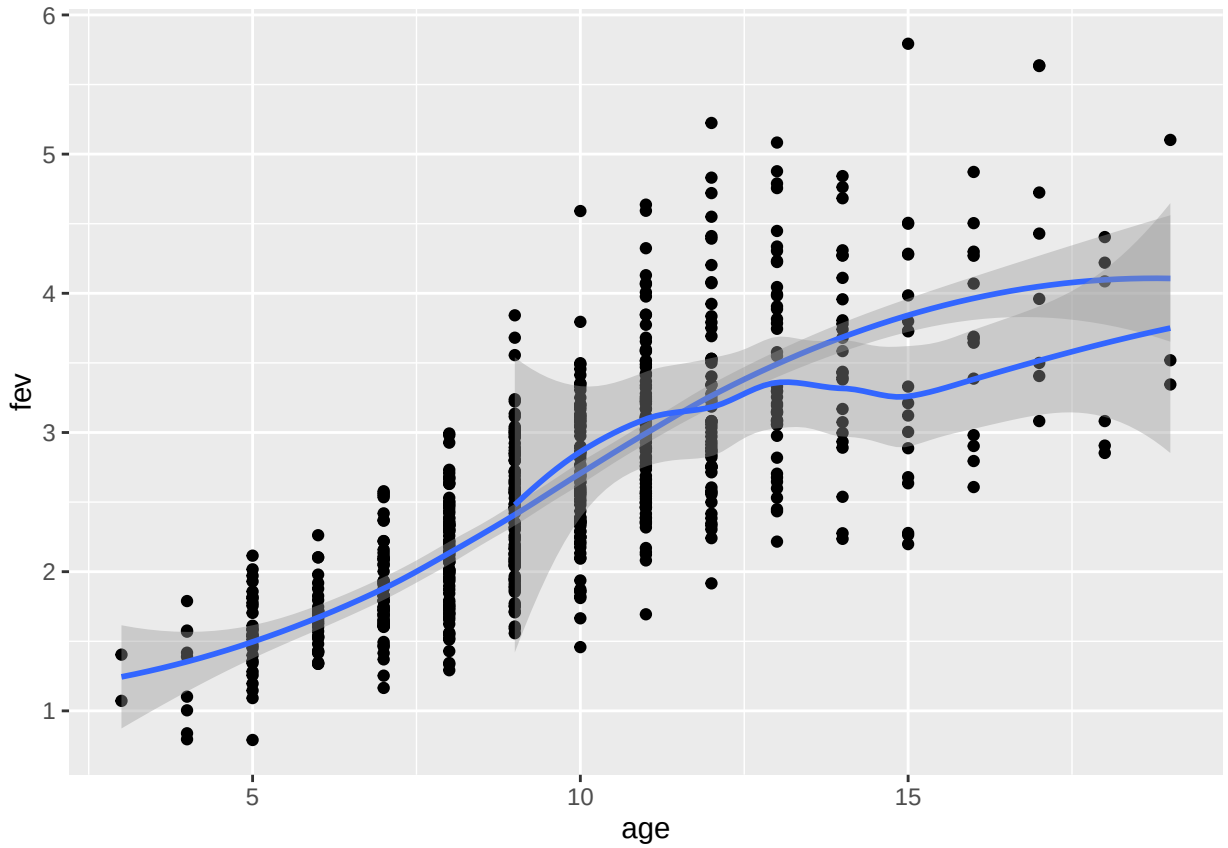
```
s + geom_point() + stat_smooth(method = "lm")
```



Grouping

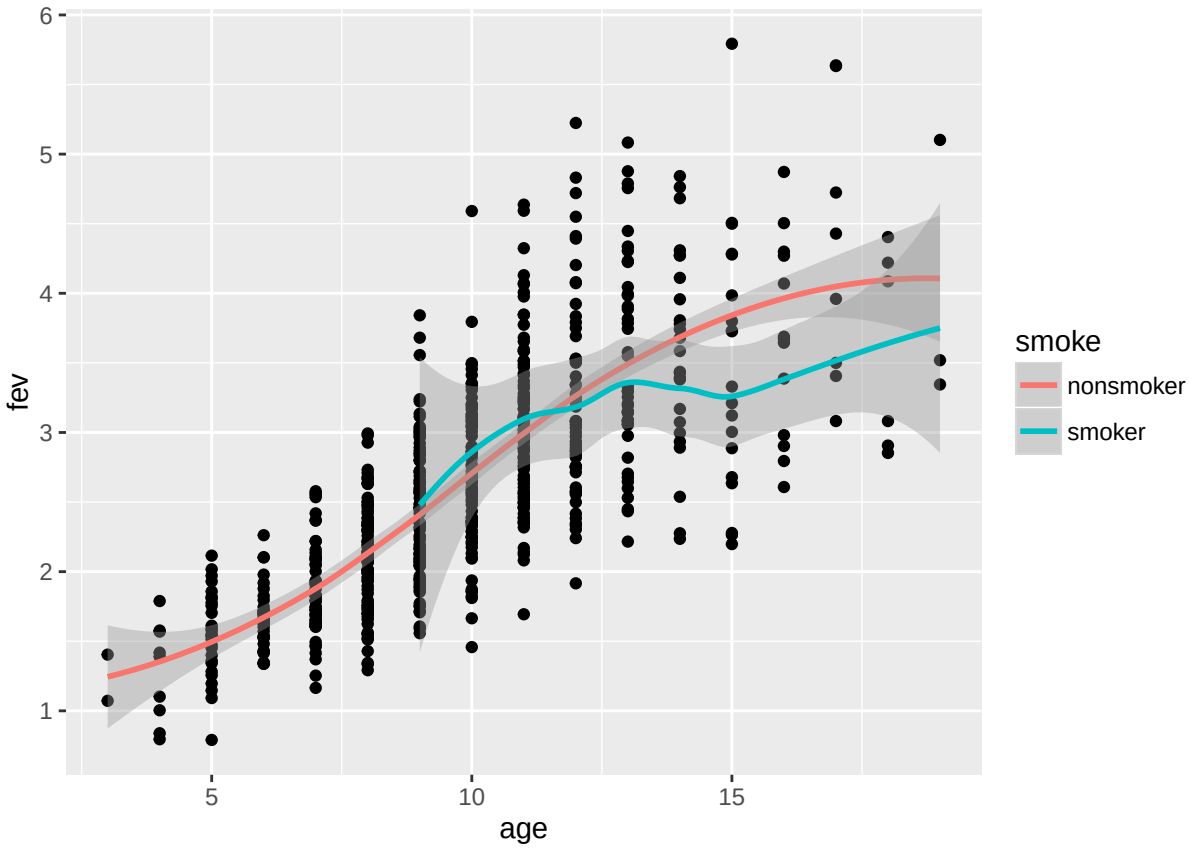
- Clearly, we can see `age` and `fev` are highly correlated. What else is correlated with `age` and `fev`?
- How can we compare the relationship between `age` and `fev` among smokers and non-smokers? One way is to use two separates smoothers: one for smokers and one for non-smokers.
- Can do this using the group aesthetic. It specifies that we wish to group the data according by some variable before layering.

```
p <- ggplot(fevdata, aes(x = age, y = fev, group = smoke))
p + geom_point() + stat_smooth()
```



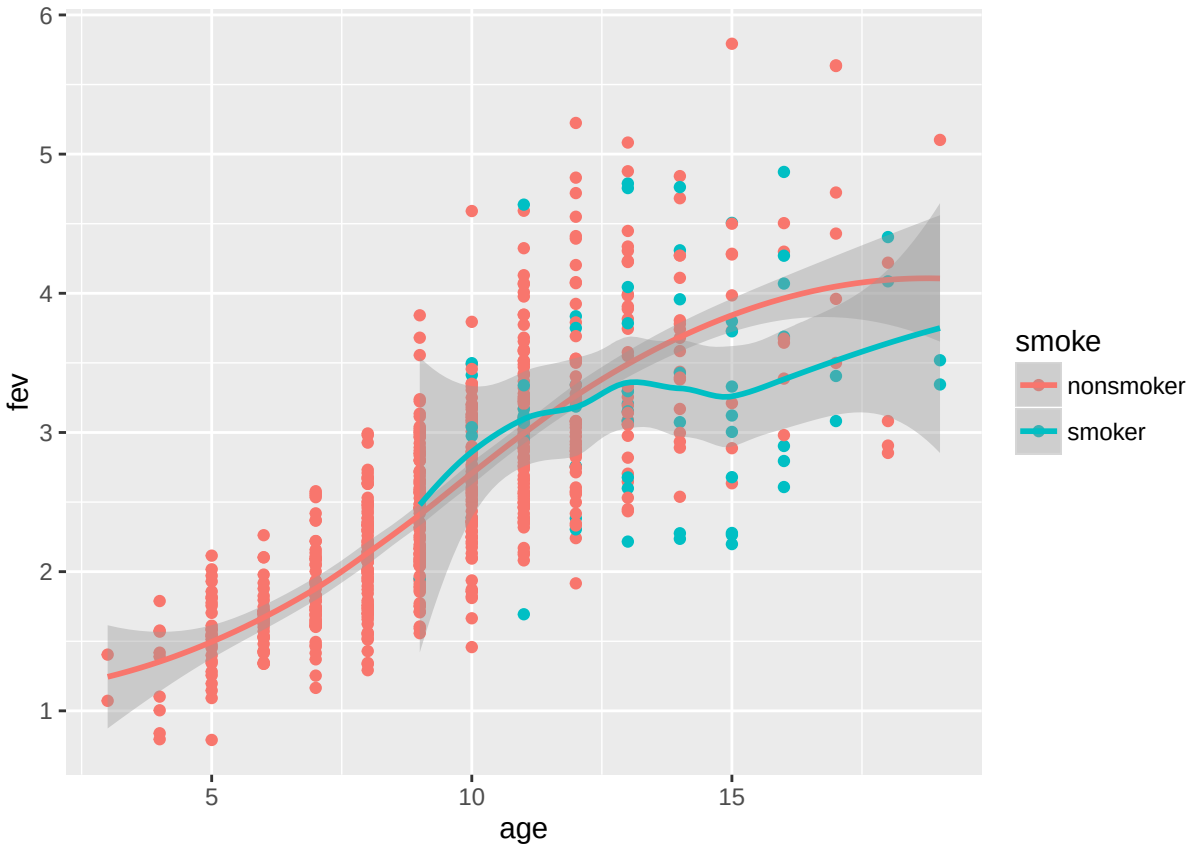
A problem with this plot is that we can't tell which group each curve corresponds to.

```
p + geom_point() + stat_smooth(aes(color = smoke))
```



Or for points and smooths

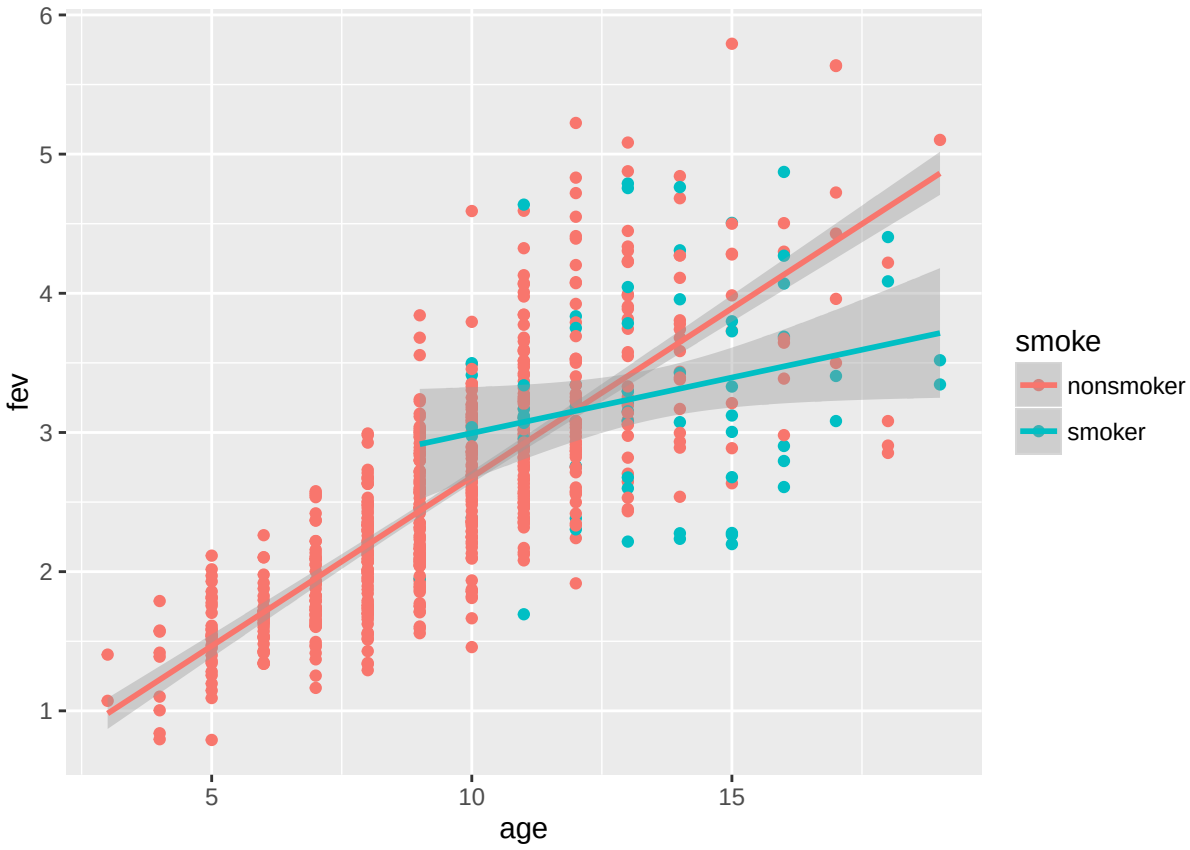
```
p <- ggplot(fevdata, aes(x = age, y = fev, group = smoke, color = smoke))
p + geom_point() + stat_smooth()
```



- What do these plots suggest?
- What can't be seen in these plots?
- Could there be lurking confounders?

How is the following plot different from the others in terms of the conclusions you might draw about the relation between `smoke` and `fev`?

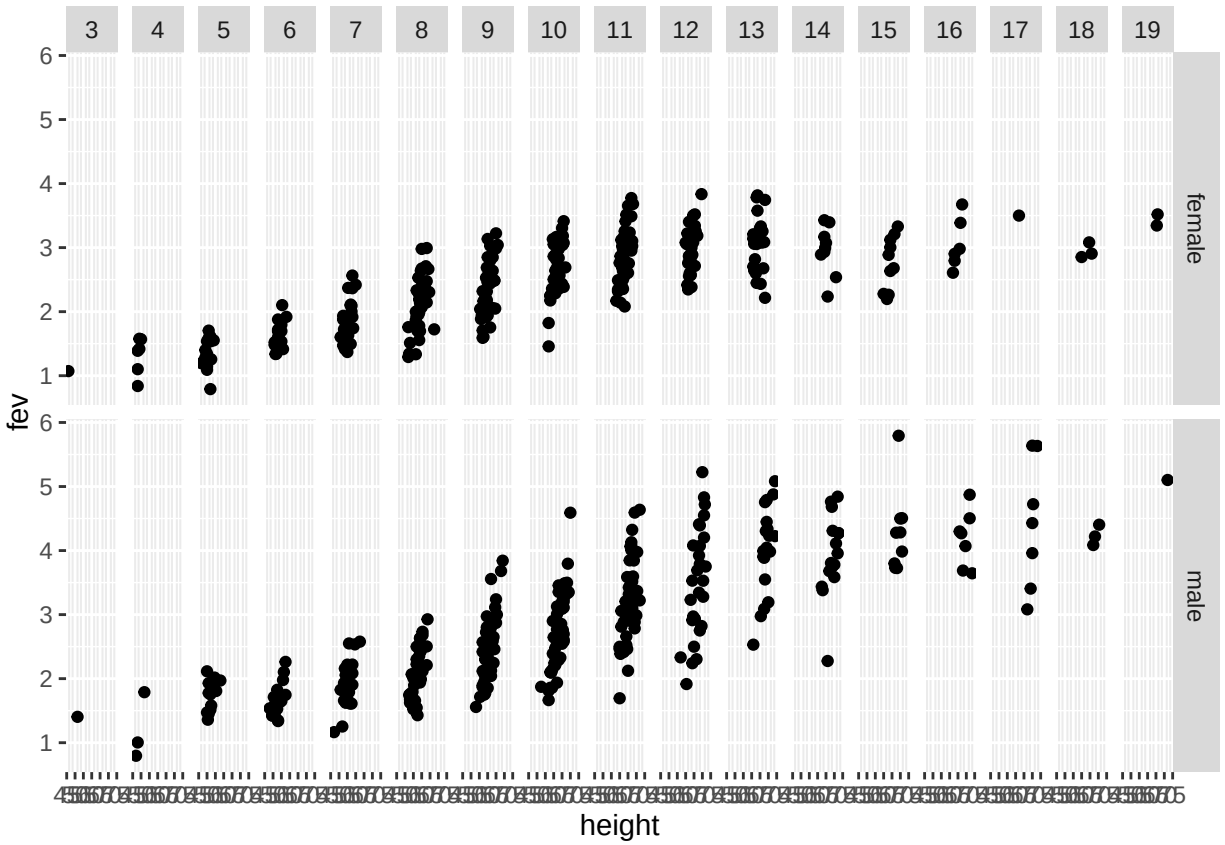
```
p + geom_point() + stat_smooth(method = "lm")
```



Faceting

Faceting is a technique for constructing multiple subplots that show different subsets of data. `ggplot2` has two ways to facet: `facet_wrap` and `facet_grid`. Grid faceting allows you use specify up to two factor variables: one for rows and one for columns of the grid. Wrap faceting allows you to specify one factor variable. We can use faceting to examine the relationship between `fev` and `height` for different combinations of `age` and `sex`. This will allow us to view four variables simultaneously.

```
p <- ggplot(fevdata, aes(x = height, y = fev)) + facet_grid(sex ~ age)
p + geom_point()
```

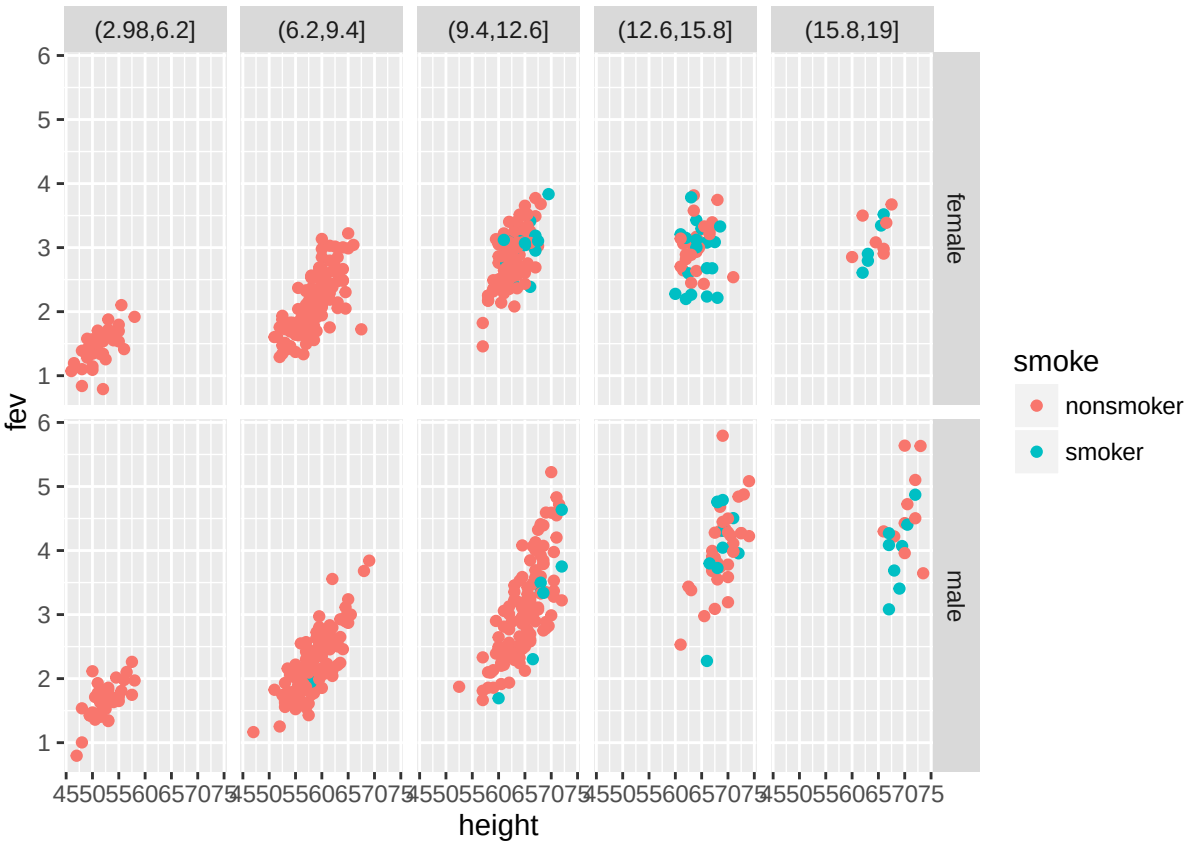


One problem with the previous plot is that there are too many ages and relatively few observations at each age. We can instead try dividing age into a smaller number of groups using the `cut()` function. `cut()` creates a new factor variable by cutting its input. Here we cut age into 5 intervals of equal length:

```
fevdata <- transform(fevdata, age_group = cut(age, breaks = 5))
```

Then make new plots:

```
p <- ggplot(fevdata, aes(x = height, y = fev)) + facet_grid(sex ~ age_group)
p + geom_point(aes(color = smoke))
```



Summary

- R has strong graphic capabilities
- Graphing is an iterative process; Don't rely on the default options
- Avoid gimmicks, use the minimum amount of ink to get your point across
- A small table can be better than a large graph
- Carefully consider the size and shape of your graph, bigger is not always better

Lecture 5: Writing Functions

Agenda

- Defining functions: Tying related commands into bundles
- Interfaces: Controlling what the function can see and do
- Example: Parameter estimation code
- Multiple functions
- Recursion: Making hard problems simpler

Why Functions?

- Data structures tie related values into one object
- Functions tie related commands into one object
- In both cases: easier to understand, easier to work with, easier to build into larger things

Example cubic function

```
cube <- function(x) x^3
cube
```

```
## function(x) x^3
```

```
cube(3)
```

```
## [1] 27
```

```
cube(1:10)
```

```
##  [1]    1    8   27   64  125  216  343  512  729 1000
```

```
cube(matrix(1:8, 2, 4))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1   27  125  343
## [2,]    8   64  216  512
```

```
matrix(cube(1:8), 2, 4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1   27  125  343
## [2,]    8   64  216  512
```

```
# cube(array(1:24, c(2, 3, 4))) # cube each element in an array
mode(cube)
```

```
## [1] "function"
```

Example

```
# 'Robust' loss function, for outlier-resistant regression Inputs: vector of
# numbers (x) Outputs: vector with x^2 for small entries, 2|x|-1 for large
# ones
```

```
psi.1 <- function(x) {
    psi <- ifelse(x^2 > 1, 2 * abs(x) - 1, x^2)
    return(psi)
}
```

Our functions get used just like the built-in ones:

```
z <- c(-0.5, -5, 0.9, 9)
psi.1(z)
```

```
## [1]  0.25  9.00  0.81 17.00
```

Go back to the declaration and look at the parts:

```
# 'Robust' loss function, for outlier-resistant regression Inputs: vector of
# numbers (x) Outputs: vector with x^2 for small entries, |x| for large ones
psi.1 <- function(x) {
    psi <- ifelse(x^2 > 1, 2 * abs(x) - 1, x^2)
    return(psi)
}
```

Interfaces: the **inputs** or **arguments**; the **outputs** or **return value**

Calls other functions `ifelse()`, `abs()`, operators `^` and `>`, and could also call other functions we've written

`return()` says what the output is; alternately, return the last evaluation

Comments: Not required by R, but a good idea

What should be a function?

- Things you're going to re-run, especially if it will be re-run with changes
- Chunks of code you keep highlighting and hitting return on
- Chunks of code which are small parts of bigger analyses
- Chunks which are very similar to other chunks

Named and default arguments

```
# 'Robust' loss function, for outlier-resistant regression Inputs: vector of
# numbers (x), scale for crossover (c) Outputs: vector with x^2 for small
# entries, 2c|x|-c^2 for large ones
psi.2 <- function(x, c = 1) {
    psi <- ifelse(x^2 > c^2, 2 * c * abs(x) - c^2, x^2)
    return(psi)
}
```

```
identical(psi.1(z), psi.2(z, c = 1))
```

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

Default values get used if names are missing:

```
identical(psi.2(z, c = 1), psi.2(z))
```

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

Named arguments can go in any order when explicitly tagged:

```
identical(psi.2(x = z, c = 2), psi.2(c = 2, x = z))
```

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

Checking Arguments

Problem: Odd behavior when arguments aren't as we expect

```
psi.2(x = z, c = c(1, 1, 1, 10))
```

```
## [1]  0.25  9.00  0.81 81.00
```

```
psi.2(x = z, c = -1)
```

```
## [1]   0.25 -11.00   0.81 -19.00
```

Solution: Put little sanity checks into the code

```
# 'Robust' loss function, for outlier-resistant regression Inputs: vector of
# numbers (x), scale for crossover (c) Outputs: vector with x^2 for small
# entries, 2c|x|-c^2 for large ones
psi.3 <- function(x, c = 1) {
    # Scale should be a single positive number
    stopifnot(length(c) == 1, c > 0)
    psi <- ifelse(x^2 > c^2, 2 * c * abs(x) - c^2, x^2)
    return(psi)
}
```

Arguments to `stopifnot()` are a series of expressions which should all be TRUE; execution halts, with error message, at *first* FALSE (try it!)

What the function can see and do

- Each function has its own environment
- Names here over-ride names in the global environment
- Internal environment starts with the named arguments
- Assignments inside the function only change the internal environment (There *are* ways around this, but they are difficult and best avoided)
- Names undefined in the function are looked for in the environment the function gets called from *not* the environment of definition

Internal environment examples

```
x <- 7
y <- c("A", "C", "G", "T", "U")
adder <- function(y) {
    x <- x + y
    return(x)
}
adder(1)
```

```
## [1] 8
```

```
x
```

```
## [1] 7
```

```
y
```

```
## [1] "A" "C" "G" "T" "U"
```

```
circle.area <- function(r) {
    return(pi * r^2)
}
circle.area(c(1, 2, 3))
```

```
## [1]  3.141593 12.566371 28.274334
```

```
truepi <- pi
pi <- 3
circle.area(c(1, 2, 3))
```

```
## [1]  3 12 27
```

```
pi <- truepi  # Restore sanity
circle.area(c(1, 2, 3))
```

```
## [1]  3.141593 12.566371 28.274334
```

Respect the interfaces

- Interfaces mark out a controlled inner environment for our code
- Interact with the rest of the system only at the interface
- Advice: arguments explicitly give the function all the information
- Reduces risk of confusion and error
- Exception: true universals like π
- Likewise, output should only be through the return value

Fitting a Model

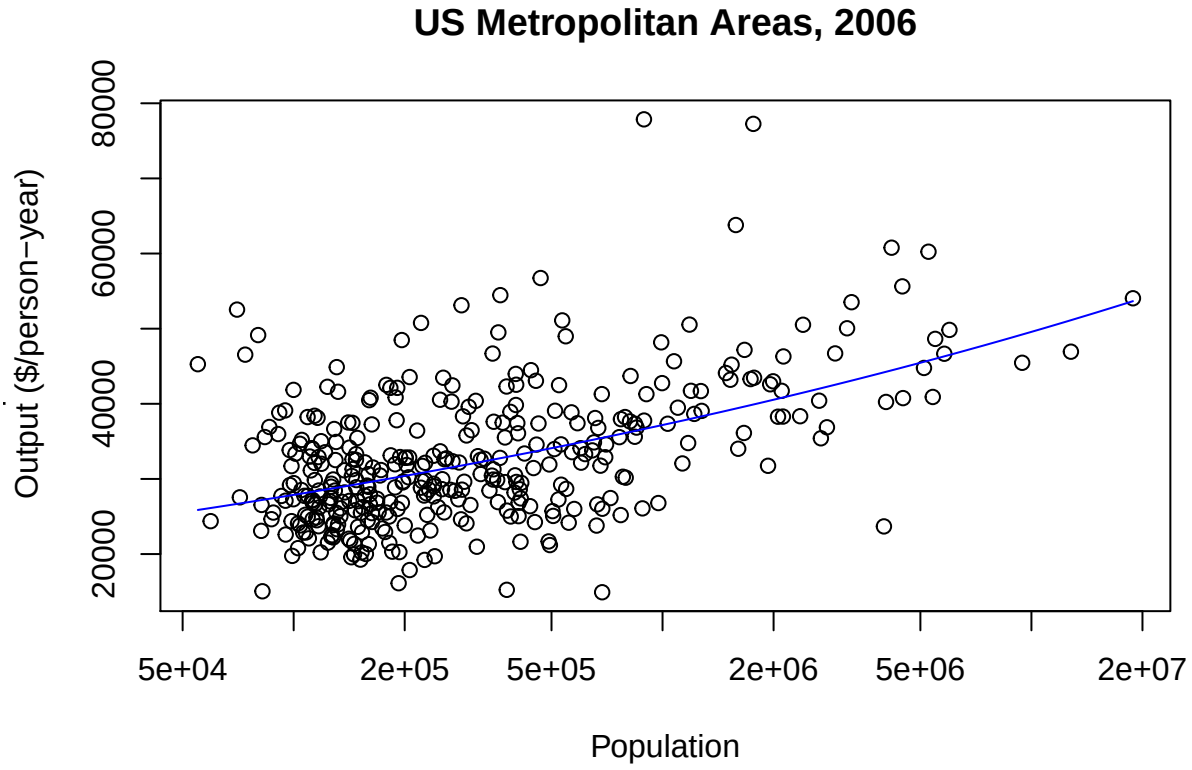
Fact: bigger cities tend to produce more economically per capita

A proposed statistical model (Geoffrey West et al.):

$$Y = y_0 N^a + \text{noise}$$

where Y is the per-capita "gross metropolitan product" of a city, N is its population, and y_0 and a are parameters

```
gmp <- read.table("http://faculty.ucr.edu/~jflegal/206/gmp.dat")
gmp$pop <- gmp$gmp/gmp$pcgmp
plot(pcgmp ~ pop, data = gmp, log = "x", xlab = "Population", ylab = "Per-Capita Economic
     Output ($/person-year)",
    main = "US Metropolitan Areas, 2006")
curve(6611 * x^(1/8), add = TRUE, col = "blue")
```



$$Y = y_0 N^a + \text{noise}$$

Take $y_0 = 6611$ for now and estimate a by minimizing the mean squared error

Approximate the derivative of error w.r.t a and move against it

$$MSE(a) \equiv \frac{1}{n}\sum_{i=1}^{n}(Y_i - y_0 N_i^a)^2 MSE'(a) \approx \frac{MSE(a+h) - MSE(a)}{h} a_{t+1} - a_t \propto -MSE'(a)$$

An actual first attempt at code:

```
maximum.iterations <- 100
deriv.step <- 1/1000
step.scale <- 1e-12
stopping.deriv <- 1/100
iteration <- 0
deriv <- Inf
a <- 0.15
while ((iteration < maximum.iterations) && (deriv > stopping.deriv)) {
    iteration <- iteration + 1
    mse.1 <- mean((gmp$pcgmp - 6611 * gmp$pop^a)^2)
    mse.2 <- mean((gmp$pcgmp - 6611 * gmp$pop^(a + deriv.step))^2)
    deriv <- (mse.2 - mse.1)/deriv.step
    a <- a - step.scale * deriv
}
list(a = a, iterations = iteration, converged = (iteration < maximum.iterations))
```

```
## $a
## [1] 0.1258166
```

```
##
## $iterations
## [1] 58
##
## $converged
## [1] TRUE
```

What's wrong with this?

- Not *encapsulated*: Re-run by cutting and pasting code — but how much of it? Also, hard to make part of something larger
- *Inflexible*: To change initial guess at a, have to edit, cut, paste, and re-run
- *Error-prone*: To change the data set, have to edit, cut, paste, re-run, and hope that all the edits are consistent
- *Hard to fix*: should stop when *absolute value* of derivative is small, but this stops when large and negative. Imagine having five copies of this and needing to fix same bug on each.

Will turn this into a function and then improve it

First attempt, with logic fix:

```
estimate.scaling.exponent.1 <- function(a) {
    maximum.iterations <- 100
    deriv.step <- 1/1000
    step.scale <- 1e-12
    stopping.deriv <- 1/100
    iteration <- 0
    deriv <- Inf
    while ((iteration < maximum.iterations) && (abs(deriv) > stopping.deriv)) {
        iteration <- iteration + 1
        mse.1 <- mean((gmp$pcgmp - 6611 * gmp$pop^a)^2)
        mse.2 <- mean((gmp$pcgmp - 6611 * gmp$pop^(a + deriv.step))^2)
        deriv <- (mse.2 - mse.1)/deriv.step
        a <- a - step.scale * deriv
    }
    fit <- list(a = a, iterations = iteration, converged = (iteration < maximum.iterations))
    return(fit)
}
```

Problem: All those magic numbers!

Solution: Make them defaults

```
estimate.scaling.exponent.2 <- function(a, y0 = 6611, maximum.iterations = 100,
    deriv.step = 0.001, step.scale = 1e-12, stopping.deriv = 0.01) {
    iteration <- 0
    deriv <- Inf
    while ((iteration < maximum.iterations) && (abs(deriv) > stopping.deriv)) {
        iteration <- iteration + 1
        mse.1 <- mean((gmp$pcgmp - y0 * gmp$pop^a)^2)
        mse.2 <- mean((gmp$pcgmp - y0 * gmp$pop^(a + deriv.step))^2)
        deriv <- (mse.2 - mse.1)/deriv.step
        a <- a - step.scale * deriv
    }
    fit <- list(a = a, iterations = iteration, converged = (iteration < maximum.iterations))
```

```
    return(fit)
}
```

Problem: Why type out the same calculation of the MSE twice?
Solution: Declare a function

```
estimate.scaling.exponent.3 <- function(a, y0 = 6611, maximum.iterations = 100,
    deriv.step = 0.001, step.scale = 1e-12, stopping.deriv = 0.01) {
    iteration <- 0
    deriv <- Inf
    mse <- function(a) {
        mean((gmp$pcgmp - y0 * gmp$pop^a)^2)
    }
    while ((iteration < maximum.iterations) && (abs(deriv) > stopping.deriv)) {
        iteration <- iteration + 1
        deriv <- (mse(a + deriv.step) - mse(a))/deriv.step
        a <- a - step.scale * deriv
    }
    fit <- list(a = a, iterations = iteration, converged = (iteration < maximum.iterations))
    return(fit)
}
```

`mse()` declared inside the function, so it can see `y0`, but it's not added to the global environment

Problem: Locked in to using specific columns of `gmp`; shouldn't have to re-write just to compare two data sets
Solution: More arguments, with defaults

```
estimate.scaling.exponent.4 <- function(a, y0 = 6611, response = gmp$pcgmp,
    predictor = gmp$pop, maximum.iterations = 100, deriv.step = 0.001, step.scale = 1e-12,
    stopping.deriv = 0.01) {
    iteration <- 0
    deriv <- Inf
    mse <- function(a) {
        mean((response - y0 * predictor^a)^2)
    }
    while ((iteration < maximum.iterations) && (abs(deriv) > stopping.deriv)) {
        iteration <- iteration + 1
        deriv <- (mse(a + deriv.step) - mse(a))/deriv.step
        a <- a - step.scale * deriv
    }
    fit <- list(a = a, iterations = iteration, converged = (iteration < maximum.iterations))
    return(fit)
}
```

Respecting the interfaces: We could turn the `while()` loop into a `for()` loop, and nothing outside the function would care

```
estimate.scaling.exponent.5 <- function(a, y0 = 6611, response = gmp$pcgmp,
    predictor = gmp$pop, maximum.iterations = 100, deriv.step = 0.001, step.scale = 1e-12,
    stopping.deriv = 0.01) {
    mse <- function(a) {
        mean((response - y0 * predictor^a)^2)
    }
    for (iteration in 1:maximum.iterations) {
        deriv <- (mse(a + deriv.step) - mse(a))/deriv.step
        a <- a - step.scale * deriv
```

```
        if (abs(deriv) <= stopping.deriv) {
            (break)()
        }
    }
    fit <- list(a = a, iterations = iteration, converged = (iteration < maximum.iterations))
    return(fit)
}
```

What have we done?

The final code is shorter, clearer, more flexible, and more re-usable

Exercise: Run the code with the default values to get an estimate of a; plot the curve along with the data points

Exercise: Randomly remove one data point — how much does the estimate change?

Exercise: Run the code from multiple starting points — how different are the estimates of a?

How We Extend Functions

- Multiple functions: Doing different things to the same object
- Sub-functions: Breaking up big jobs into small ones

Why Multiple Functions?

Meta-problems:

- You've got more than one problem
- Your problem is too hard to solve in one step
- You keep solving the same problems

Meta-solutions:

- Write multiple functions, which rely on each other
- Split your problem, and write functions for the pieces
- Solve the recurring problems once, and re-use the solutions

Writing Multiple Related Functions

Statisticians want to do lots of things with their models: estimate, predict, visualize, test, compare, simulate, uncertainty, ...

Write multiple functions to do these things

Make the model one object; assume it has certain components

Consistent Interfaces

- Functions for the same kind of object should use the same arguments, and presume the same structure
- Functions for the same kind of task should use the same arguments, and return the same sort of value (to the extent possible)

Keep related things together

- Put all the related functions in a single file
- Source them together
- Use comments to note ***dependencies***

Power-Law Scaling

Remember the model:

$$Y = y_0 N^a + \text{noise}$$

$$\text{(output per person)} =$$

$$\text{(baseline)(population)}^{\text{scaling exponent}} + \text{noise}$$

Estimated parameters a, y_0 by minimizing the mean squared error

Exercise: Modify the estimation code from last time so it returns a list, with components `a` and `y0`

Predicting from a Fitted Model

Predict values from the power-law model:

```
# Predict response values from a power-law scaling model Inputs: fitted
# power-law model (object), vector of values at which to make predictions at
# (newdata) Outputs: vector of predicted response values
predict.plm <- function(object, newdata) {
    # Check that object has the right components
    stopifnot("a" %in% names(object), "y0" %in% names(object))
    a <- object$a
    y0 <- object$y0
    # Sanity check the inputs
    stopifnot(is.numeric(a), length(a) == 1)
    stopifnot(is.numeric(y0), length(y0) == 1)
    stopifnot(is.numeric(newdata))
    return(y0 * newdata^a)  # Actual calculation and return
}
```

```
# Plot fitted curve from power law model over specified range Inputs: list
# containing parameters (plm), start and end of range (from, to) Outputs:
# TRUE, silently, if successful Side-effect: Makes the plot
plot.plm.1 <- function(plm, from, to) {
    # Take sanity-checking of parameters as read
    y0 <- plm$y0  # Extract parameters
    a <- plm$a
    f <- function(x) {
        return(y0 * x^a)
    }
    curve(f(x), from = from, to = to)
    # Return with no visible value on the terminal
    invisible(TRUE)
}
```

Predicting from a Fitted Model

When one function calls another, use `...` as a meta-argument, to pass along unspecified inputs to the called function:

```
plot.plm.2 <- function(plm, ...) {
    y0 <- plm$y0
    a <- plm$a
    f <- function(x) {
        return(y0 * x^a)
    }
    # from and to are possible arguments to curve()
    curve(f(x), ...)
    invisible(TRUE)
}
```

Sub-Functions

Solve big problems by dividing them into a few sub-problems

- Easier to understand, get the big picture at a glance
- Easier to fix, improve and modify
- Easier to design
- Easier to re-use solutions to recurring sub-problems

Rule of thumb: A function longer than a page is probably too long

Sub-Functions or Separate Functions?

Defining a function inside another function

- Pros: Simpler code, access to local variables, doesn't clutter workspace
- Cons: Gets re-declared each time, can't access in global environment (or in other functions)
- Alternative: Declare the function in the same file, source them together

Rule of thumb: If you find yourself writing the same code in multiple places, make it a separate function

Plotting a Power-Law Model

Our old plotting function calculated the fitted values

But so does our prediction function

```
plot.plm.3 <- function(plm, from, to, n = 101, ...) {
    x <- seq(from = from, to = to, length.out = n)
    y <- predict.plm(object = plm, newdata = x)
    plot(x, y, ...)
    invisible(TRUE)
}
```

Recursion

Reduce the problem to an easier one of the same form:

```
my.factorial <- function(n) {
    if (n == 1) {
        return(1)
    } else {
        return(n * my.factorial(n - 1))
    }
}
```

Or multiple calls (Fibonacci numbers):

```
fib <- function(n) {
    if ((n == 1) || (n == 0)) {
        return(1)
    } else {
        return(fib(n - 1) + fib(n - 2))
    }
}
```

Exercise: Convince yourself that any loop can be replaced by recursion; can you always replace recursion with a loop?

Summary

- **Functions** bundle related commands together into objects: easier to re-run, easier to re-use, easier to combine, easier to modify, less risk of error, easier to think about
- **Interfaces** control what the function can see (arguments, environment) and change (its internals, its return value)
- **Calling** functions we define works just like calling built-in functions: named arguments, defaults
- **Multiple functions** let us do multiple related jobs, either on the same object or on similar ones
- **Sub-functions** let us break big problems into smaller ones, and re-use the solutions to the smaller ones
- Recursion is a powerful way of making hard problems simpler

Lecture 6: Getting Data and Linear Models

Agenda

- Getting data into and out of R
- Using data frames for statistical purposes
- Introduction to linear models

Reading Data from R

- You can load and save R objects
  - R has its own format for this, which is shared across operating systems
  - It's an open, documented format if you really want to pry into it
- `save(thing, file="name")` saves `thing` in a file called `name` (conventional extension: `rda` or `Rda`)
- `load("name")` loads the object or objects stored in the file called `name`, *with their old names*

```
gmp <- read.table("http://faculty.ucr.edu/~jflegal/206/gmp.dat")
gmp$pop <- round(gmp$gmp/gmp$pcgmp)
save(gmp, file = "gmp.Rda")
rm(gmp)
exists("gmp")
```

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

```
not_gmp <- load(file = "gmp.Rda")
colnames(gmp)
```

```
## [1] "MSA"   "gmp"   "pcgmp" "pop"
```

```
not_gmp
```

```
## [1] "gmp"
```

- We can load or save more than one object at once; this is how RStudio will load your whole workspace when you're starting, and offer to save it when you're done
- Many packages come with saved data objects; there's the convenience function `data()` to load them

```
data(cats, package = "MASS")
summary(cats)
```

```
##  Sex         Bwt             Hwt
##  F:47   Min.   :2.000   Min.   : 6.30
##  M:97   1st Qu.:2.300   1st Qu.: 8.95
##         Median :2.700   Median :10.10
##         Mean   :2.724   Mean   :10.63
##         3rd Qu.:3.025   3rd Qu.:12.12
##         Max.   :3.900   Max.   :20.50
```

Non-R Data Tables

- Tables full of data, just not in the R file format
- Main function: `read.table()`
  - Presumes space-separated fields, one line per row
  - Main argument is the file name or URL
  - Returns a dataframe

  - Lots of options for things like field separator, column names, forcing or guessing column types, skipping lines at the start of the file...
- `read.csv()` is a short-cut to set the options for reading comma-separated value (CSV) files
  - Spreadsheets will usually read and write CSV

Writing Dataframes

- Counterpart functions `write.table()`, `write.csv()` write a dataframe into a file
- Drawback: takes a lot more disk space than what you get from `load` or `save`
- Advantage: can communicate with other programs, or even edit manually

Less Friendly Data Formats

- The `foreign` package on CRAN has tools for reading data files from lots of non-R statistical software
- Spreadsheets are special
- Full of ugly irregularities
- Values or formulas?
- Headers, footers, side-comments, notes
- Columns change meaning half-way down

Spreadsheets, If You Have To

- Save the spreadsheet as a CSV; `read.csv()`
- Save the spreadsheet as a CSV; edit in a text editor; `read.csv()`
- Use `read.xls()` from the `gdata` package
- Tries very hard to work like `read.csv()`, can take a URL or filename
- Can skip down to the first line that matches some pattern, select different sheets, etc.
- You may still need to do a lot of tidying up after

So You've Got A Data Frame

What can we do with it?

- Plot it: examine multiple variables and distributions
- Test it: compare groups of individuals to each other
- Check it: does it conform to what we'd like for our needs

Test Case: Birth weight data

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
data(birthwt)
summary(birthwt)
```

```
##       low              age              lwt             race       
##  Min.   :0.0000   Min.   :14.00   Min.   : 80.0   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:19.00   1st Qu.:110.0   1st Qu.:1.000  
##  Median :0.0000   Median :23.00   Median :121.0   Median :1.000  
##  Mean   :0.3122   Mean   :23.24   Mean   :129.8   Mean   :1.847  
##  3rd Qu.:1.0000   3rd Qu.:26.00   3rd Qu.:140.0   3rd Qu.:3.000  
##  Max.   :1.0000   Max.   :45.00   Max.   :250.0   Max.   :3.000  
##      smoke            ptl               ht                ui        
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.0000  
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.0000  
##  Median :0.0000   Median :0.0000   Median :0.00000   Median :0.0000  
##  Mean   :0.3915   Mean   :0.1958   Mean   :0.06349   Mean   :0.1481  
##  3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:0.00000   3rd Qu.:0.0000  
##  Max.   :1.0000   Max.   :3.0000   Max.   :1.00000   Max.   :1.0000  
##       ftv              bwt        
##  Min.   :0.0000   Min.   : 709  
##  1st Qu.:0.0000   1st Qu.:2414  
##  Median :0.0000   Median :2977  
##  Mean   :0.7937   Mean   :2945  
##  3rd Qu.:1.0000   3rd Qu.:3487  
##  Max.   :6.0000   Max.   :4990
```

From R help

Go to R help for more info, because someone documented this data

```
help(birthwt)
```

Make it Readable

```
colnames(birthwt)
```

```
##  [1] "low"   "age"   "lwt"   "race"  "smoke" "ptl"   "ht"    "ui"   
##  [9] "ftv"   "bwt"
```

```
colnames(birthwt) <- c("birthwt.below.2500", "mother.age", "mother.weight",
    "race", "mother.smokes", "previous.prem.labor", "hypertension", "uterine.irr",
    "physician.visits", "birthwt.grams")
```

Can make all the factors more descriptive.

```
birthwt$race <- factor(c("white", "black", "other")[birthwt$race])
birthwt$mother.smokes <- factor(c("No", "Yes")[birthwt$mother.smokes + 1])
birthwt$uterine.irr <- factor(c("No", "Yes")[birthwt$uterine.irr + 1])
birthwt$hypertension <- factor(c("No", "Yes")[birthwt$hypertension + 1])
```

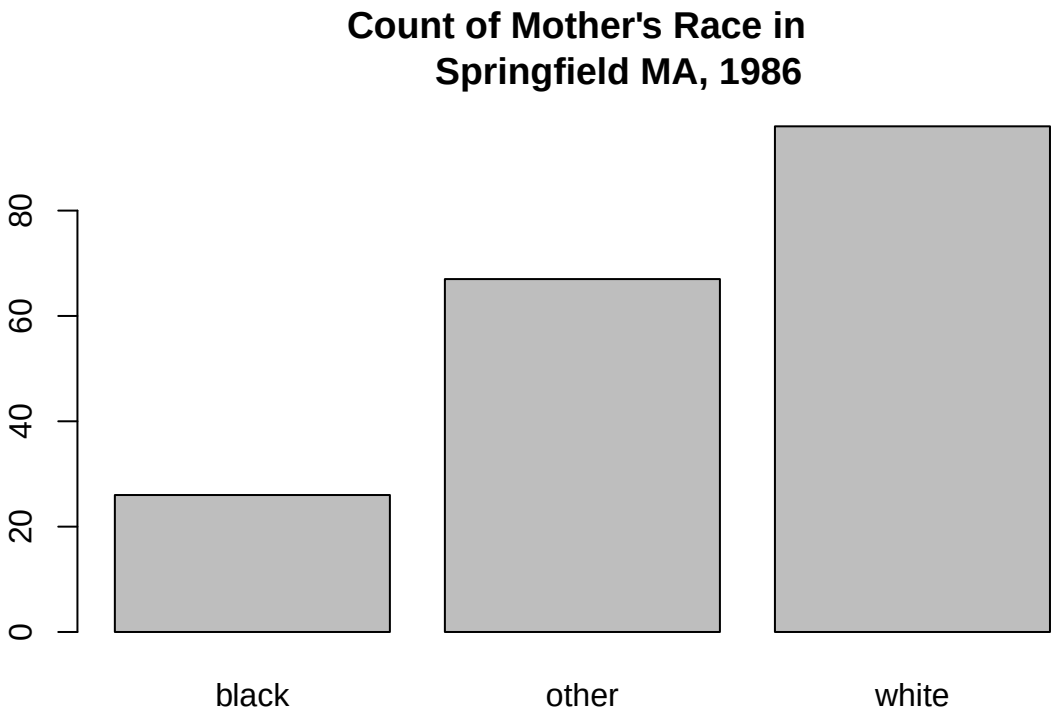
```
summary(birthwt)
```

```
##  birthwt.below.2500   mother.age    mother.weight      race    
##  Min.   :0.0000     Min.   :14.00   Min.   : 80.0   black:26  
##  1st Qu.:0.0000     1st Qu.:19.00   1st Qu.:110.0   other:67  
##  Median :0.0000     Median :23.00   Median :121.0   white:96  
##  Mean   :0.3122     Mean   :23.24   Mean   :129.8             
##  3rd Qu.:1.0000     3rd Qu.:26.00   3rd Qu.:140.0
```

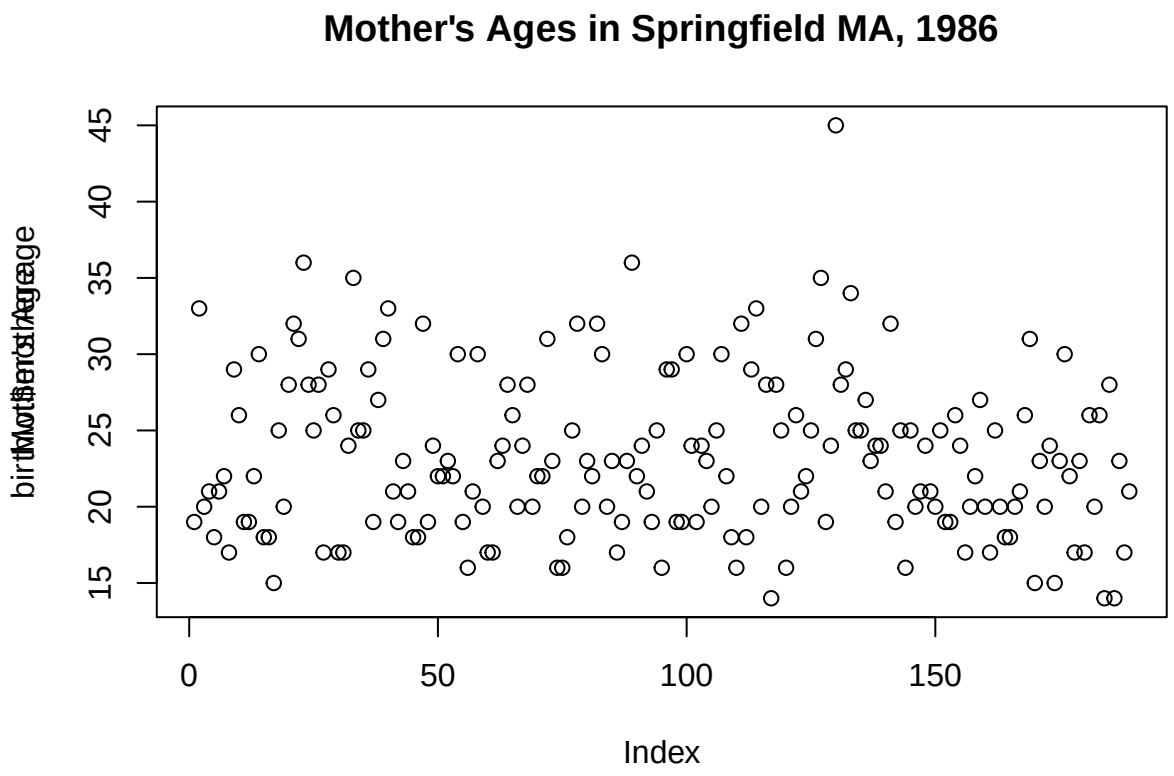
```
##  Max.   :1.0000     Max.   :45.00   Max.   :250.0
##  mother.smokes previous.prem.labor hypertension uterine.irr
##  No :115       Min.   :0.0000      No :177      No :161
##  Yes: 74       1st Qu.:0.0000      Yes: 12      Yes: 28
##                Median :0.0000
##                Mean   :0.1958
##                3rd Qu.:0.0000
##                Max.   :3.0000
##  physician.visits birthwt.grams
##  Min.   :0.0000   Min.   : 709
##  1st Qu.:0.0000   1st Qu.:2414
##  Median :0.0000   Median :2977
##  Mean   :0.7937   Mean   :2945
##  3rd Qu.:1.0000   3rd Qu.:3487
##  Max.   :6.0000   Max.   :4990
```

Explore It

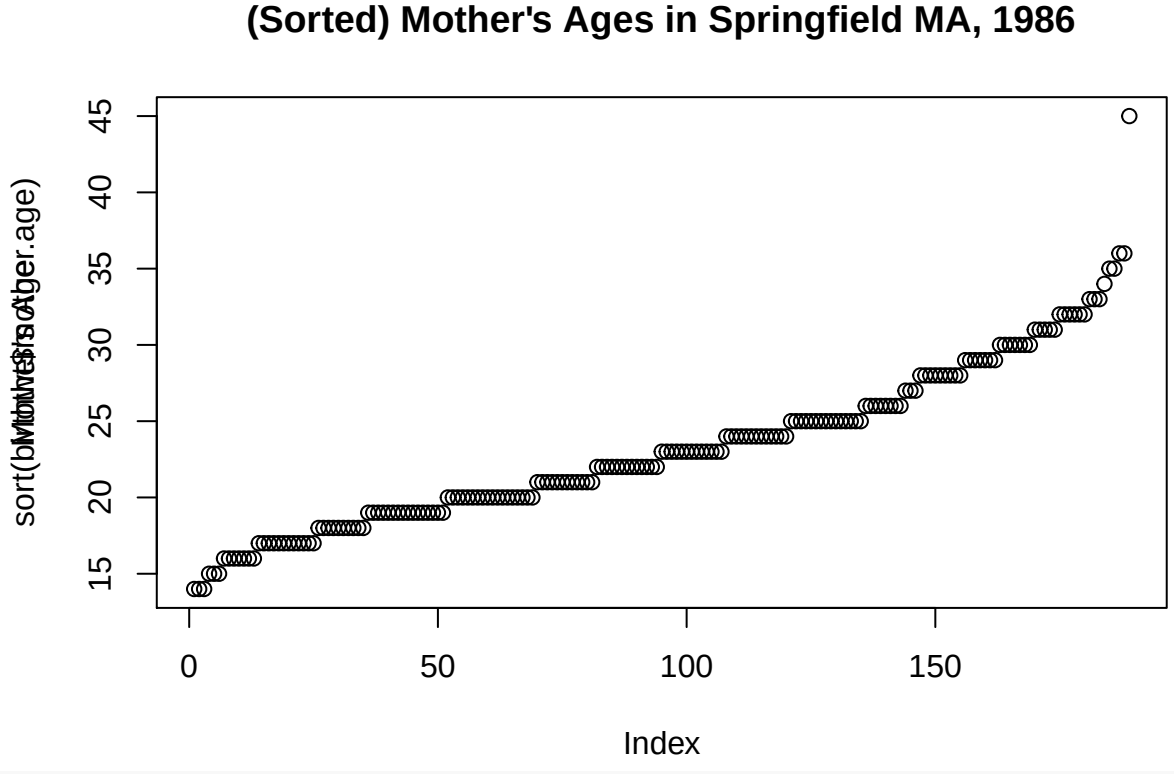
```
plot(birthwt$race)
title(main = "Count of Mother's Race in
       Springfield MA, 1986")
```



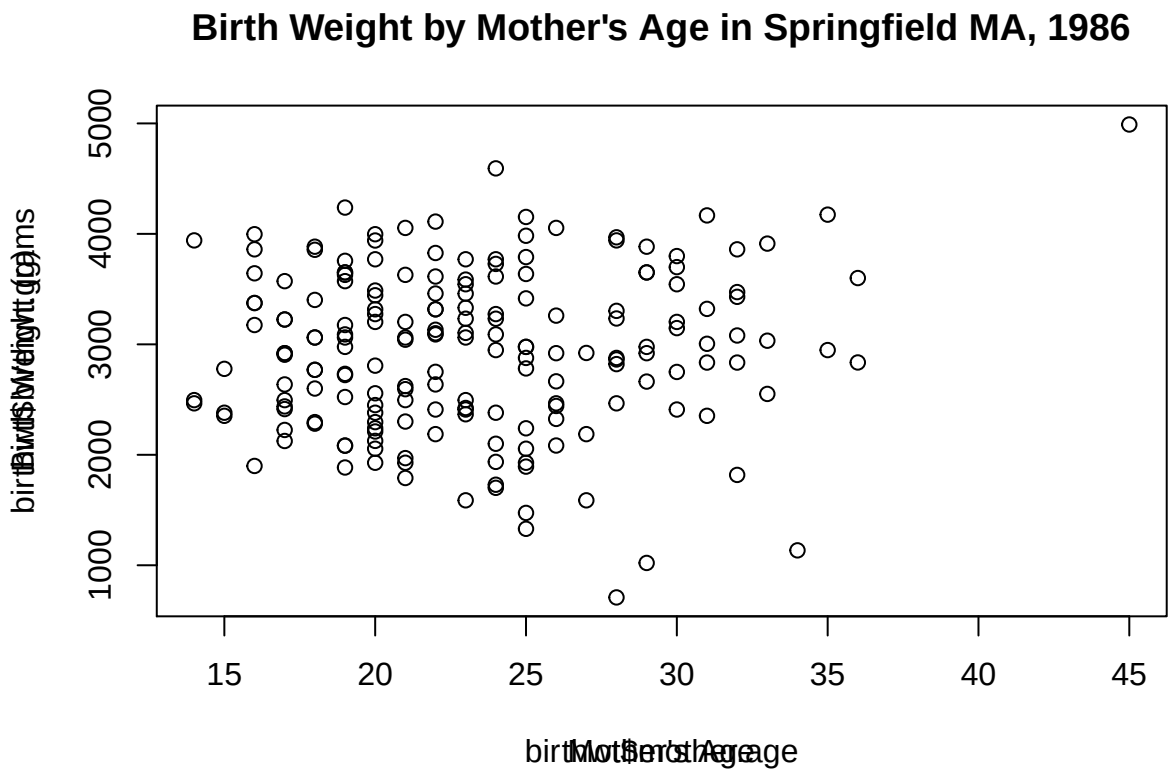

```
plot(birthwt$mother.age)
title(main = "Mother's Ages in Springfield MA, 1986", ylab = "Mother's Age")
```




```
plot(sort(birthwt$mother.age))
title(main = "(Sorted) Mother's Ages in Springfield MA, 1986", ylab = "Mother's Age")
```



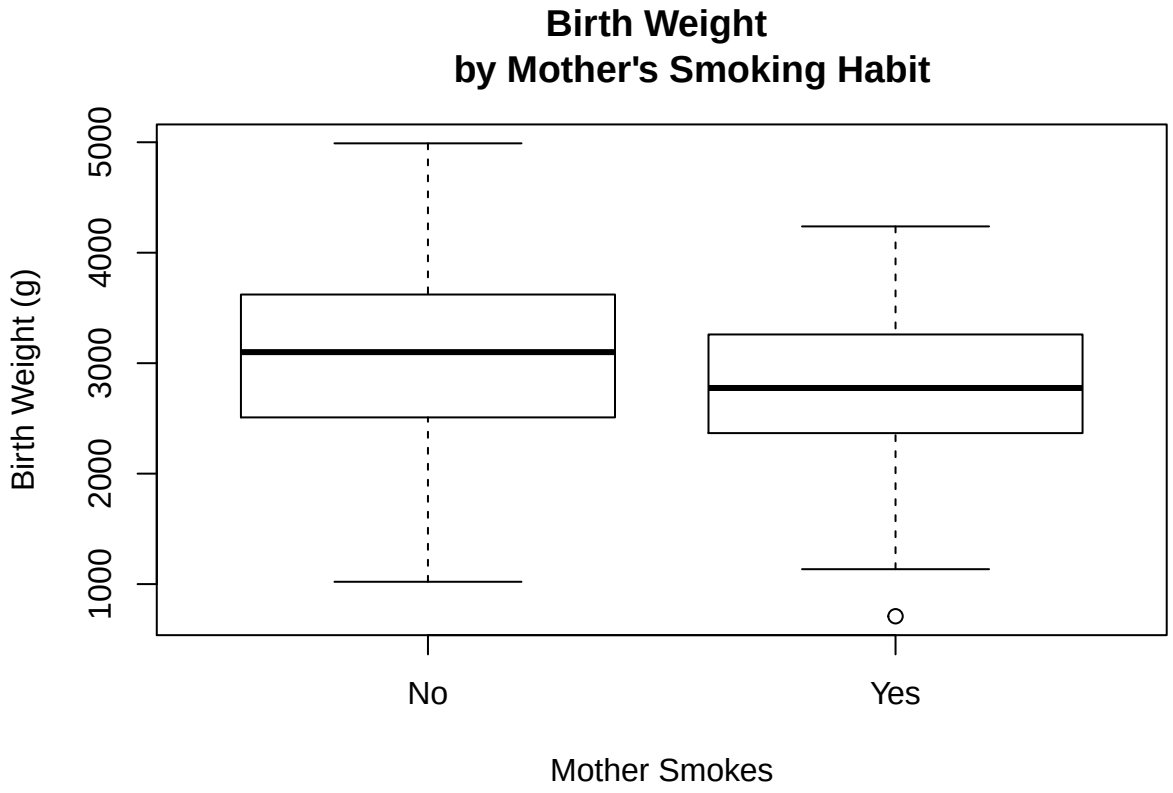

```
plot(birthwt$mother.age, birthwt$birthwt.grams)
title(main = "Birth Weight by Mother's Age in Springfield MA, 1986", xlab = "Mother's Age",
    ylab = "Birth Weight (g)")
```



Basic statistical testing

Let's fit some models to the data pertaining to our outcome(s) of interest.

```
plot(birthwt$mother.smokes, birthwt$birthwt.grams, main = "Birth Weight
      by Mother's Smoking Habit",
    ylab = "Birth Weight (g)", xlab = "Mother Smokes")
```



Tough to tell! Simple two-sample t-test:

```
t.test(birthwt$birthwt.grams[birthwt$mother.smokes == "Yes"], birthwt$birthwt.grams[birthwt$mother.smok
    "No"])
```

```
##
##  Welch Two Sample t-test
##
## data:  birthwt$birthwt.grams[birthwt$mother.smokes == "Yes"] and birthwt$birthwt.grams[birthwt$mothe
## t = -2.7299, df = 170.1, p-value = 0.007003
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -488.97860  -78.57486
## sample estimates:
## mean of x mean of y
##  2771.919  3055.696
```

Does this difference match the linear model?

```
linear.model.1 <- lm(birthwt.grams ~ mother.smokes, data = birthwt)
linear.model.1
```

```
##
## Call:
## lm(formula = birthwt.grams ~ mother.smokes, data = birthwt)
##
## Coefficients:
##      (Intercept)  mother.smokesYes
##           3055.7            -283.8
```

Does this difference match the linear model?

```
summary(linear.model.1)
```

```
##
## Call:
## lm(formula = birthwt.grams ~ mother.smokes, data = birthwt)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2062.9  -475.9    34.3   545.1  1934.3
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       3055.70      66.93  45.653  < 2e-16 ***
## mother.smokesYes  -283.78     106.97  -2.653  0.00867 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 717.8 on 187 degrees of freedom
## Multiple R-squared:  0.03627,    Adjusted R-squared:  0.03112
## F-statistic: 7.038 on 1 and 187 DF,  p-value: 0.008667
```

Does this difference match the linear model?

```
linear.model.2 <- lm(birthwt.grams ~ mother.age, data = birthwt)
linear.model.2
```

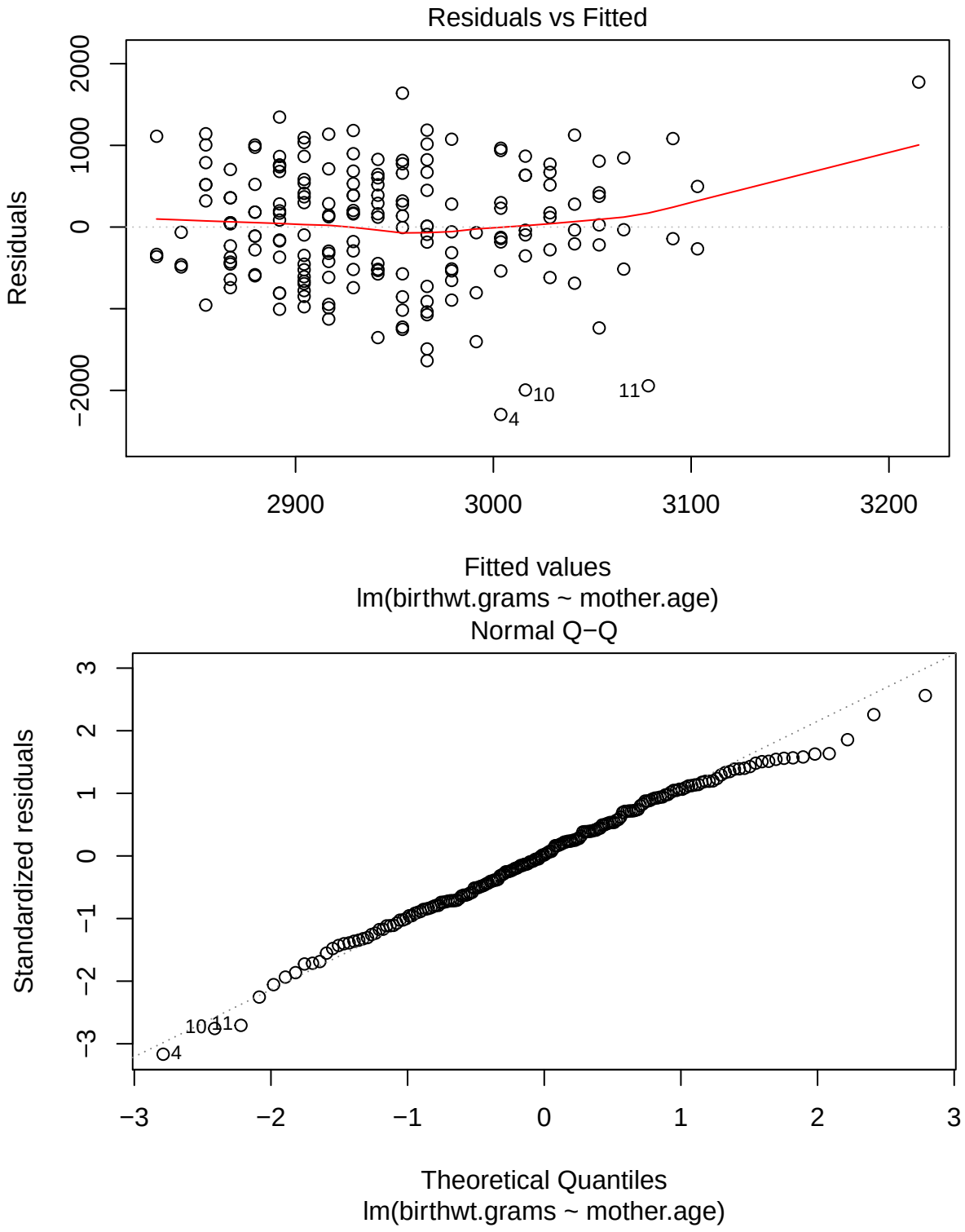
```
##
## Call:
## lm(formula = birthwt.grams ~ mother.age, data = birthwt)
##
## Coefficients:
## (Intercept)   mother.age
##     2655.74        12.43
```

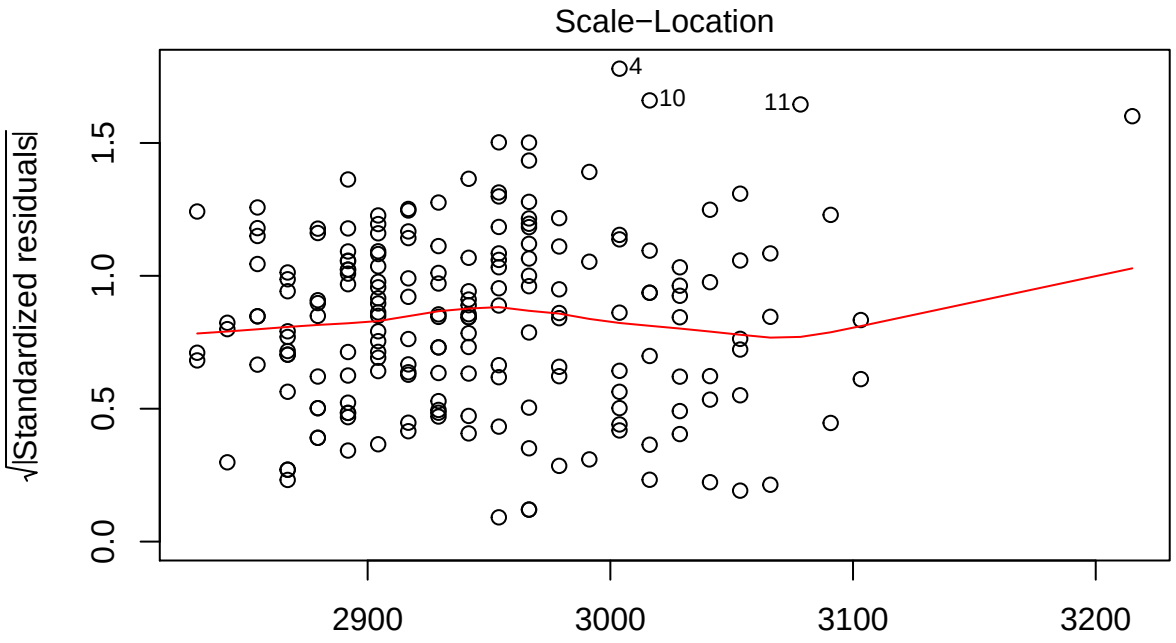
```
summary(linear.model.2)
```

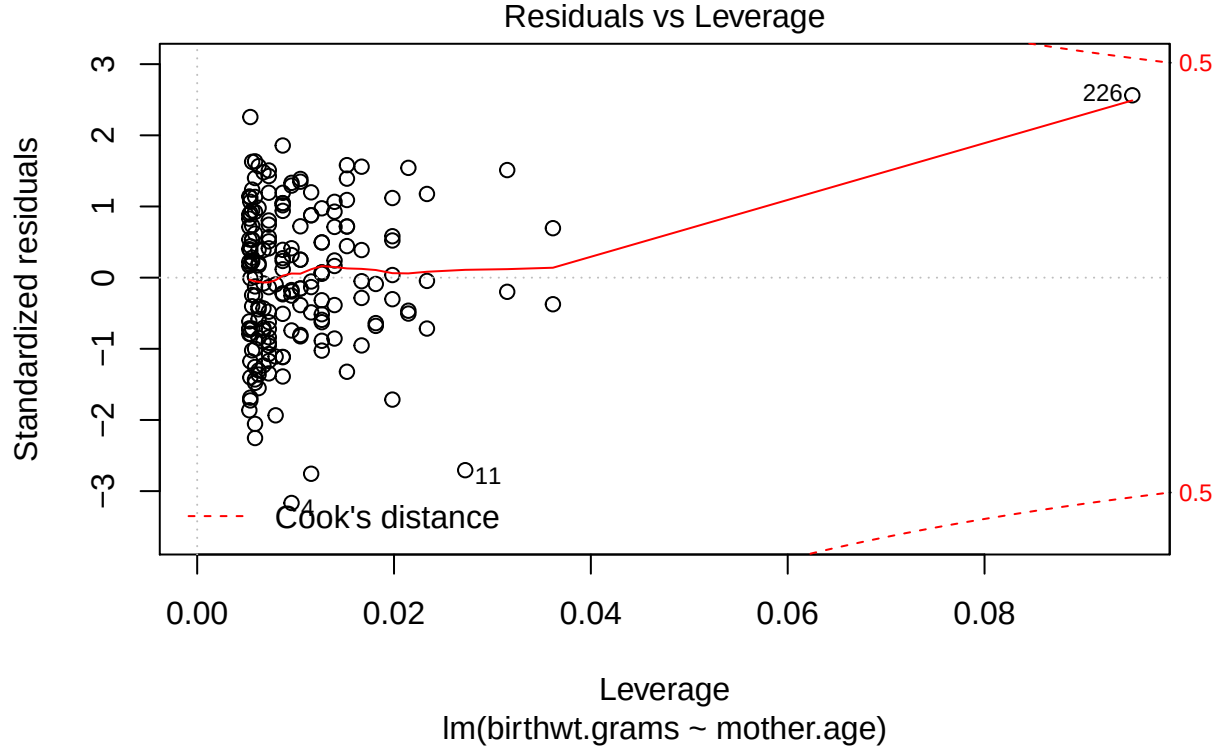
```
##
## Call:
## lm(formula = birthwt.grams ~ mother.age, data = birthwt)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2294.78  -517.63    10.51   530.80  1774.92
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2655.74     238.86   11.12   <2e-16 ***
## mother.age     12.43      10.02    1.24    0.216
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 728.2 on 187 degrees of freedom
## Multiple R-squared:  0.008157,   Adjusted R-squared:  0.002853
## F-statistic: 1.538 on 1 and 187 DF,  p-value: 0.2165
```

R tries to make diagnostics easy as possible. Try in R console.

```
plot(linear.model.2)
```







Detecting Outliers

Note the oldest mother and her heaviest child are greatly skewing this analysis.

```r
birthwt.noout <- birthwt[birthwt$mother.age <= 40, ]
linear.model.3 <- lm(birthwt.grams ~ mother.age, data = birthwt.noout)
linear.model.3
```

```
##
## Call:
## lm(formula = birthwt.grams ~ mother.age, data = birthwt.noout)
##
## Coefficients:
## (Intercept)   mother.age
##    2833.273        4.344
```

```r
summary(linear.model.3)
```

```
##
## Call:
## lm(formula = birthwt.grams ~ mother.age, data = birthwt.noout)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2245.89  -511.24    26.45   540.09  1655.48
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2833.273    244.954   11.57   <2e-16 ***
## mother.age     4.344     10.349    0.42    0.675
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 717.2 on 186 degrees of freedom
## Multiple R-squared:  0.0009461,  Adjusted R-squared:  -0.004425
## F-statistic: 0.1761 on 1 and 186 DF,  p-value: 0.6752
```

More complex models

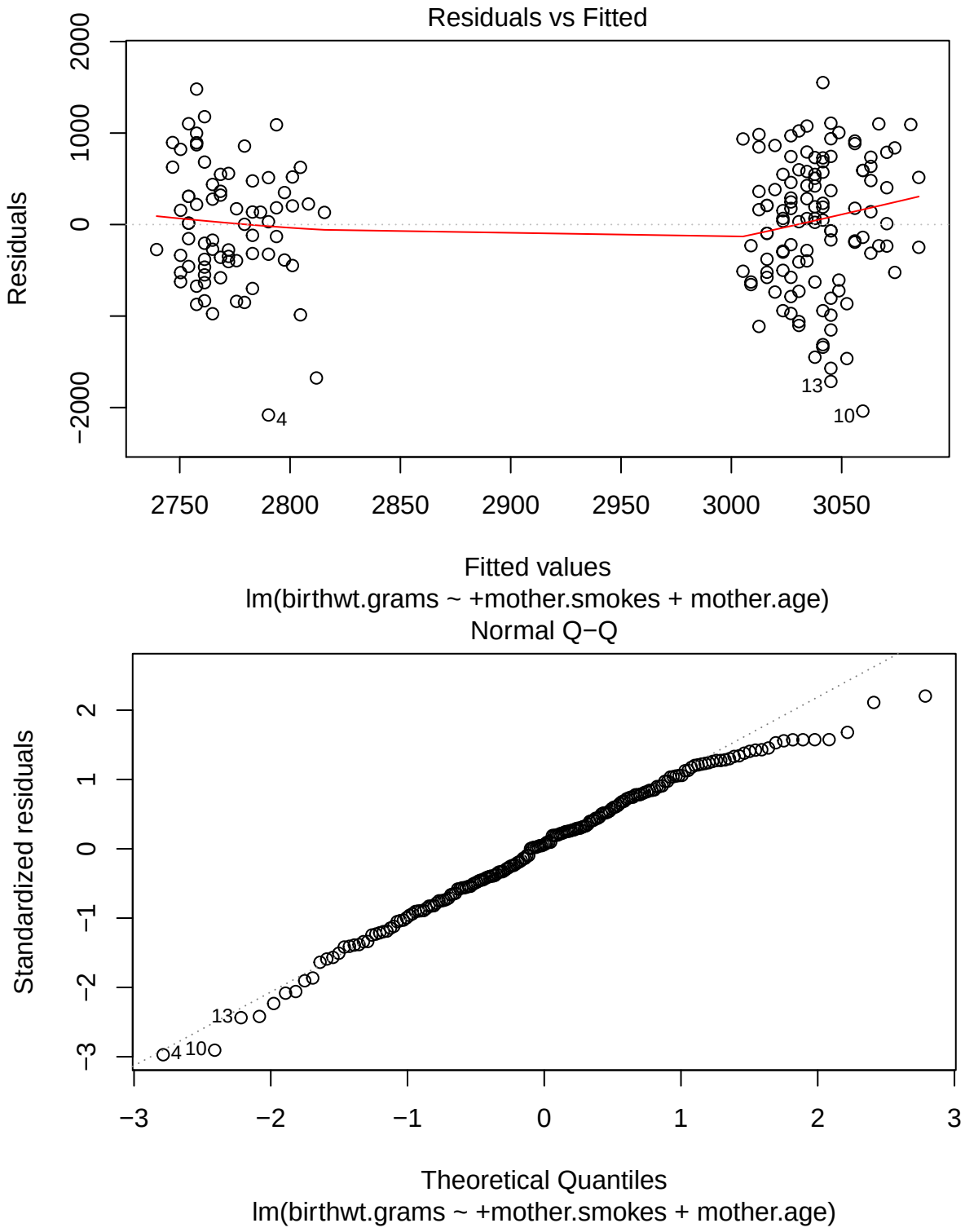
Add in smoking behavior:

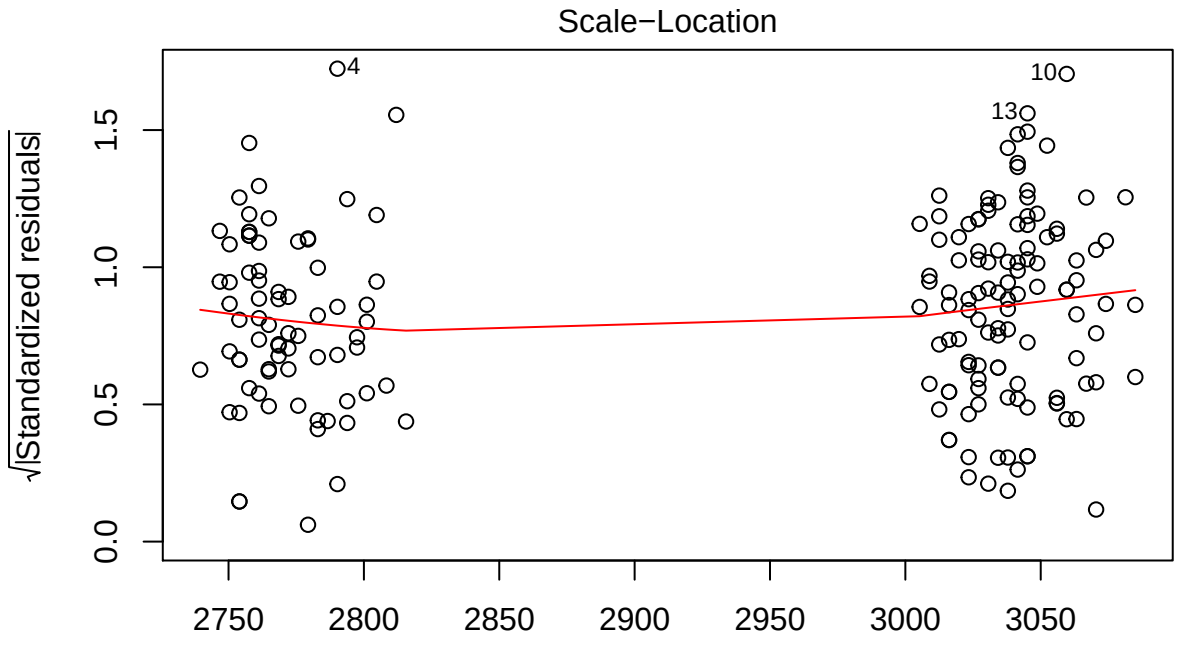
```r
linear.model.3a <- lm(birthwt.grams ~ +mother.smokes + mother.age, data = birthwt.noout)
summary(linear.model.3a)
```

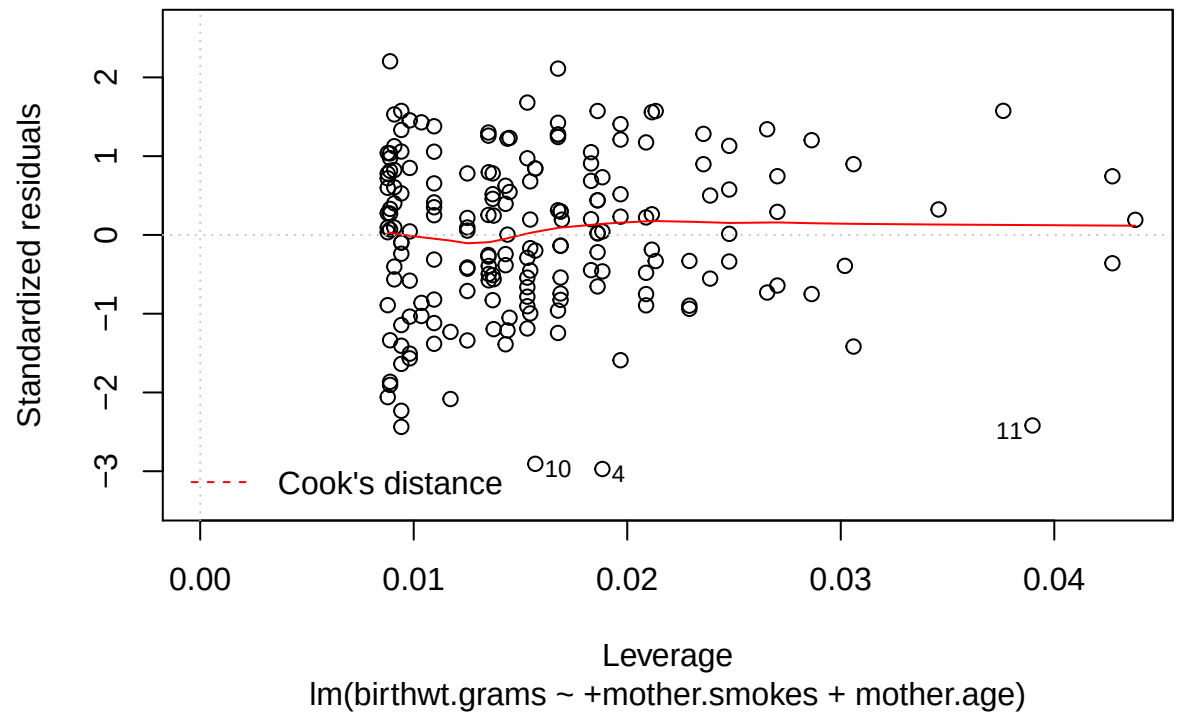
```
##
## Call:
## lm(formula = birthwt.grams ~ +mother.smokes + mother.age, data = birthwt.noout)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2081.22  -459.82    43.56   548.22  1551.51
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       2954.582    246.280  11.997   <2e-16 ***
## mother.smokesYes -265.756    105.605  -2.517   0.0127 *
## mother.age          3.621     10.208   0.355   0.7232
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 707.1 on 185 degrees of freedom
## Multiple R-squared:  0.03401,    Adjusted R-squared:  0.02357
## F-statistic: 3.257 on 2 and 185 DF,  p-value: 0.04072
```

```
plot(linear.model.3a)
```


Residuals vs Fitted
Residuals
2000
1000
0
−2000
2750
2800
2850
2900
2950
3000
3050
4
13
10
Fitted values
lm(birthwt.grams ~ +mother.smokes + mother.age)
Normal Q−Q
Standardized residuals
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
13
4
10
Theoretical Quantiles
lm(birthwt.grams ~ +mother.smokes + mother.age)





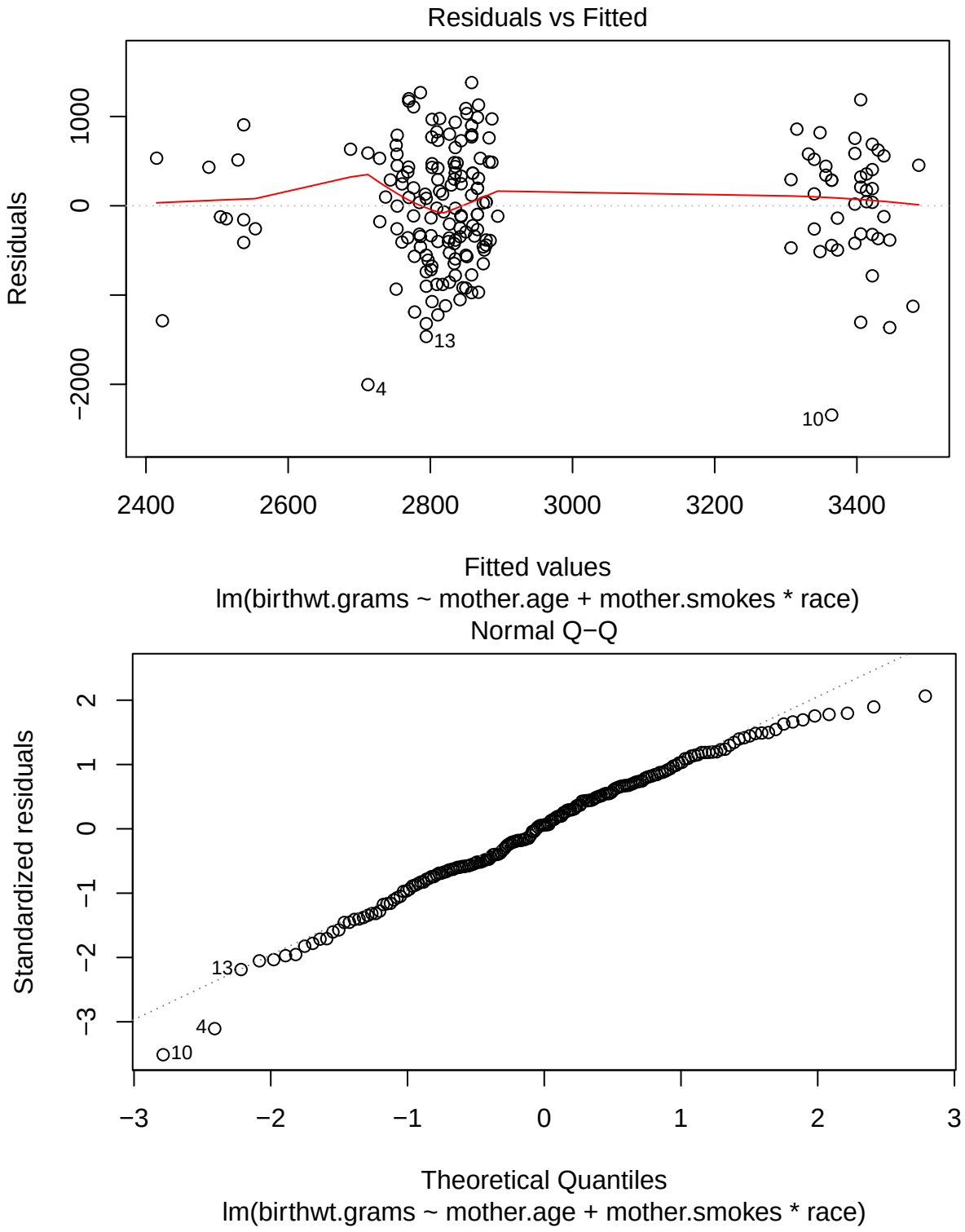
Add in race:

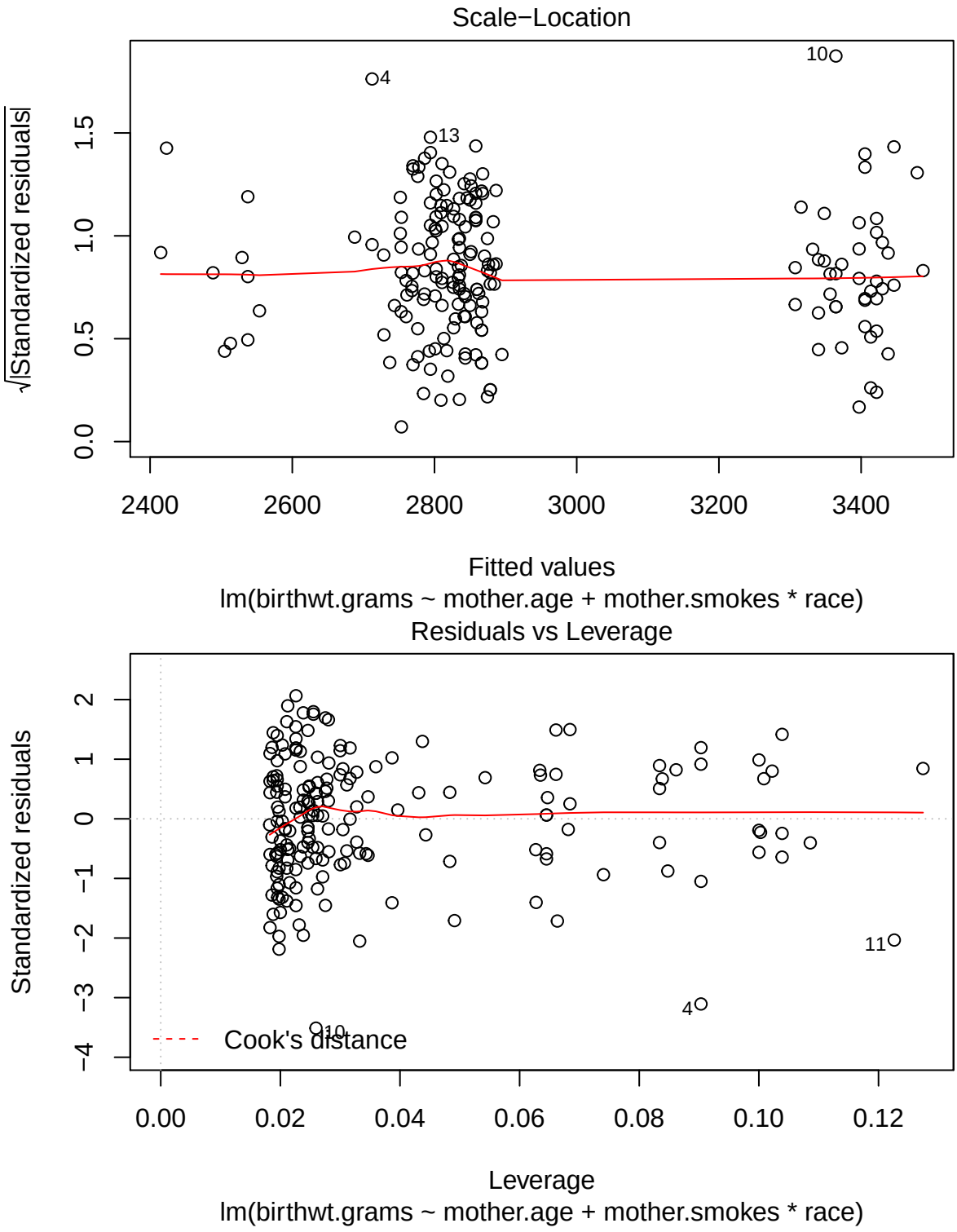
```
linear.model.3b <- lm(birthwt.grams ~ mother.age + mother.smokes * race, data = birthwt.noout)
summary(linear.model.3b)
```

```
##
## Call:
```

```
## lm(formula = birthwt.grams ~ mother.age + mother.smokes * race,
##     data = birthwt.noout)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2343.52  -413.66    39.91   480.36  1379.90
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 3017.352    265.606  11.360  < 2e-16 ***
## mother.age                    -8.168     10.276  -0.795  0.42772
## mother.smokesYes            -316.500    275.896  -1.147  0.25282
## raceother                    -18.901    193.665  -0.098  0.92236
## racewhite                    584.042    206.320   2.831  0.00517 **
## mother.smokesYes:raceother   258.999    349.871   0.740  0.46010
## mother.smokesYes:racewhite -271.594    314.268  -0.864  0.38862
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 676.1 on 181 degrees of freedom
## Multiple R-squared:  0.1359, Adjusted R-squared:  0.1073
## F-statistic: 4.746 on 6 and 181 DF,  p-value: 0.0001625
```

```
plot(linear.model.3b)
```


Residuals vs Fitted
Residuals
1000
0
−2000
2400
2600
2800
3000
3200
3400
13
4
10
Fitted values
lm(birthwt.grams ~ mother.age + mother.smokes * race)
Normal Q−Q
Standardized residuals
2
1
0
−1
−2
−3
13
4
10
−3
−2
−1
0
1
2
3
Theoretical Quantiles
lm(birthwt.grams ~ mother.age + mother.smokes * race)



Including everything

Let's include everything on this new data set:

```
linear.model.4 <- lm(birthwt.grams ~ ., data = birthwt.noout)
linear.model.4
```

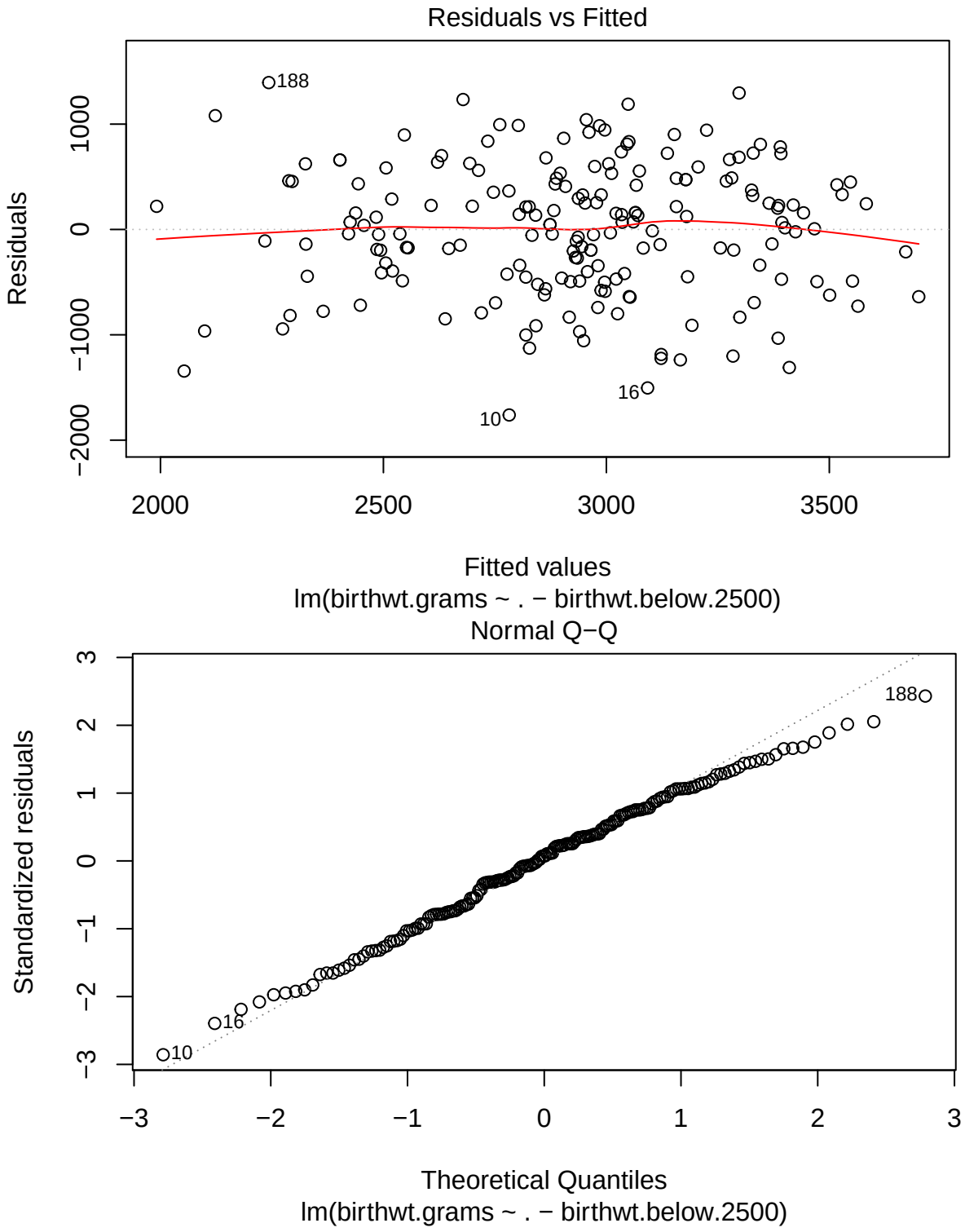
```
##
## Call:
## lm(formula = birthwt.grams ~ ., data = birthwt.noout)
##
## Coefficients:
##         (Intercept)   birthwt.below.2500           mother.age
##           3360.5163           -1116.3933              -16.0321
##       mother.weight            raceother            racewhite
##              1.9317              68.8145             247.0241
##    mother.smokesYes  previous.prem.labor      hypertensionYes
##           -157.7041              95.9825             -185.2778
##      uterine.irrYes     physician.visits
##           -340.0918              -0.3519
```

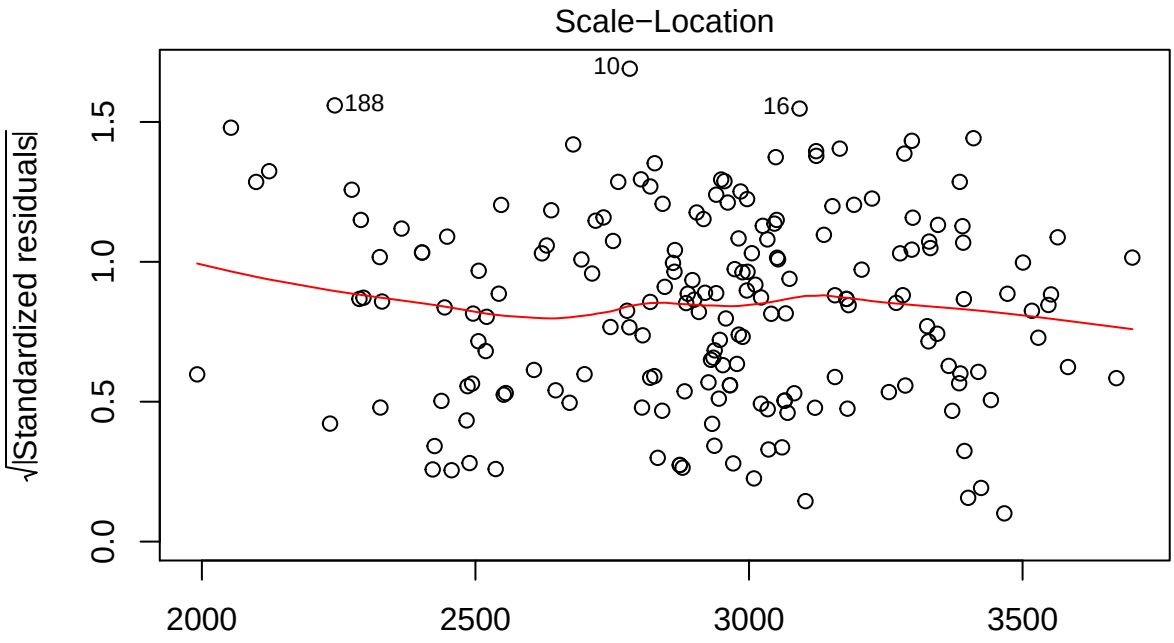
Be careful! One of those variables `birthwt.below.2500` is a function of the outcome.

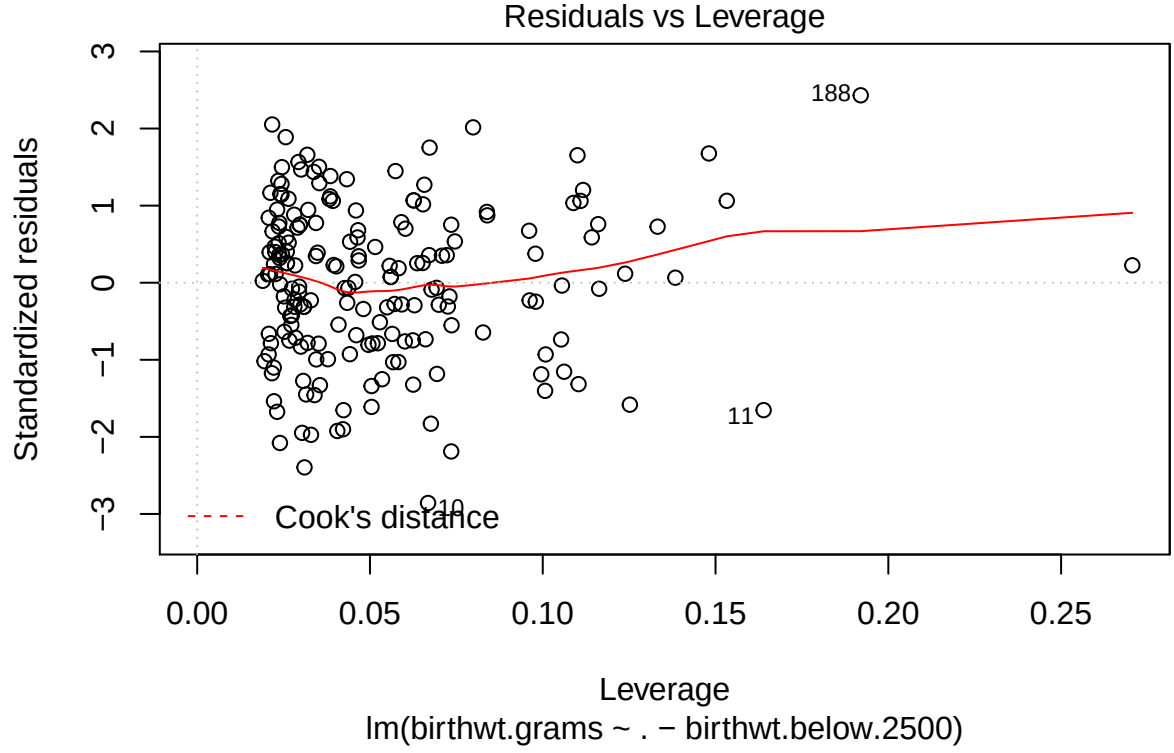
```
linear.model.4a <- lm(birthwt.grams ~ . - birthwt.below.2500, data = birthwt.noout)
summary(linear.model.4a)
```

```
##
## Call:
## lm(formula = birthwt.grams ~ . - birthwt.below.2500, data = birthwt.noout)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1761.10  -454.81    46.43   459.78  1394.13
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)         2545.584    323.204   7.876 3.21e-13 ***
## mother.age           -12.111      9.909  -1.222 0.223243
## mother.weight          4.789      1.710   2.801 0.005656 **
## raceother            155.605    156.564   0.994 0.321634
## racewhite            494.545    147.153   3.361 0.000951 ***
## mother.smokesYes    -335.793    104.613  -3.210 0.001576 **
## previous.prem.labor  -32.922    100.185  -0.329 0.742838
## hypertensionYes     -594.324    198.480  -2.994 0.003142 **
## uterine.irrYes      -514.842    136.249  -3.779 0.000215 ***
## physician.visits      -7.247     45.649  -0.159 0.874036
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 638 on 178 degrees of freedom
## Multiple R-squared:  0.2435, Adjusted R-squared:  0.2052
## F-statistic: 6.365 on 9 and 178 DF,  p-value: 8.255e-08
```

```
plot(linear.model.4a)
```


Residuals vs Fitted
Residuals
Fitted values
lm(birthwt.grams ~ . − birthwt.below.2500)
188
16
10
Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(birthwt.grams ~ . − birthwt.below.2500)

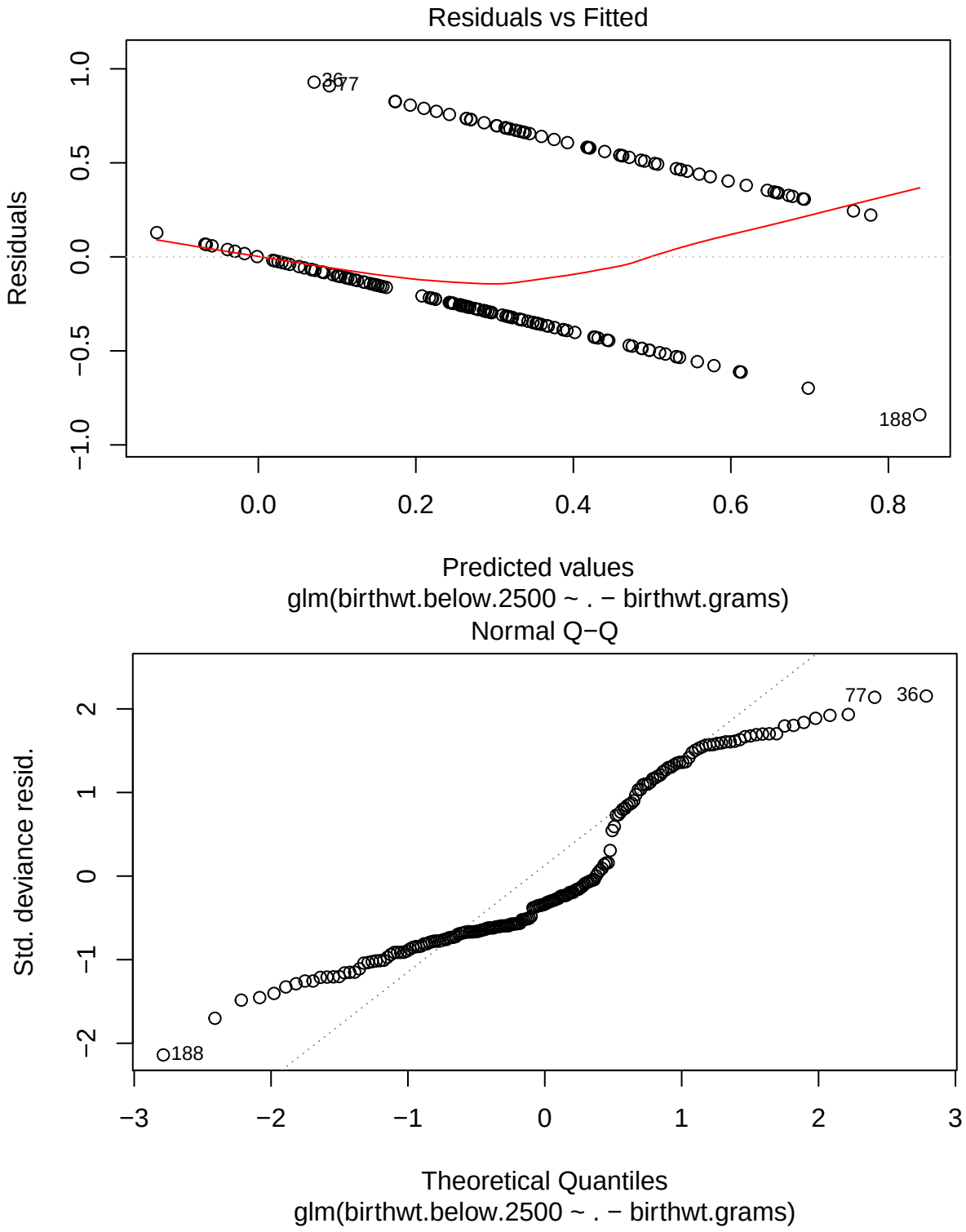


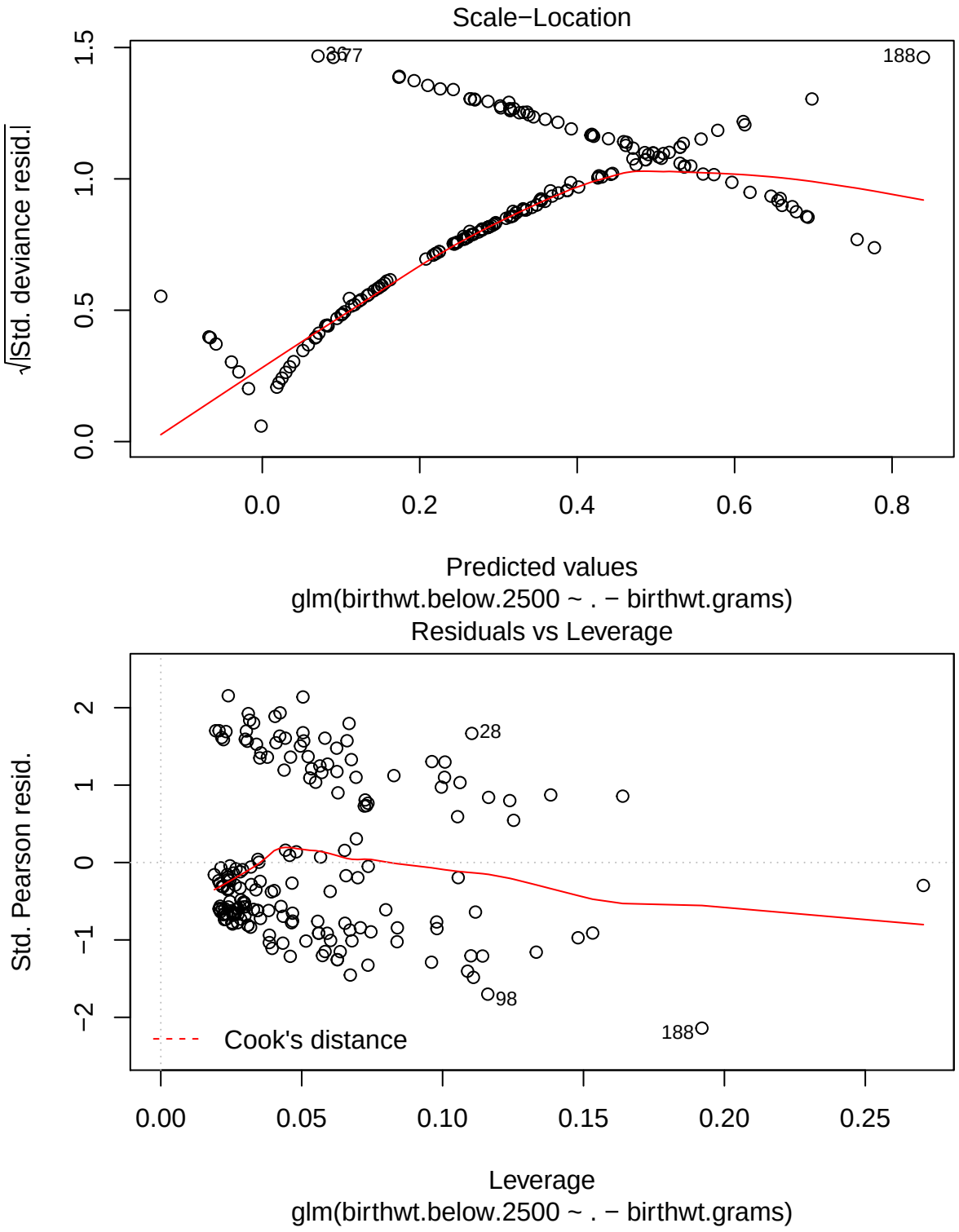


Generalized Linear Models

Maybe a linear increase in birth weight is less important than if it's below a threshold like 2500 grams (5.5 pounds). Let's fit a generalized linear model instead:

```
glm.0 <- glm(birthwt.below.2500 ~ . - birthwt.grams, data = birthwt.noout)
plot(glm.0)
```





The default value is a Gaussian model (a standard linear model). Change this:

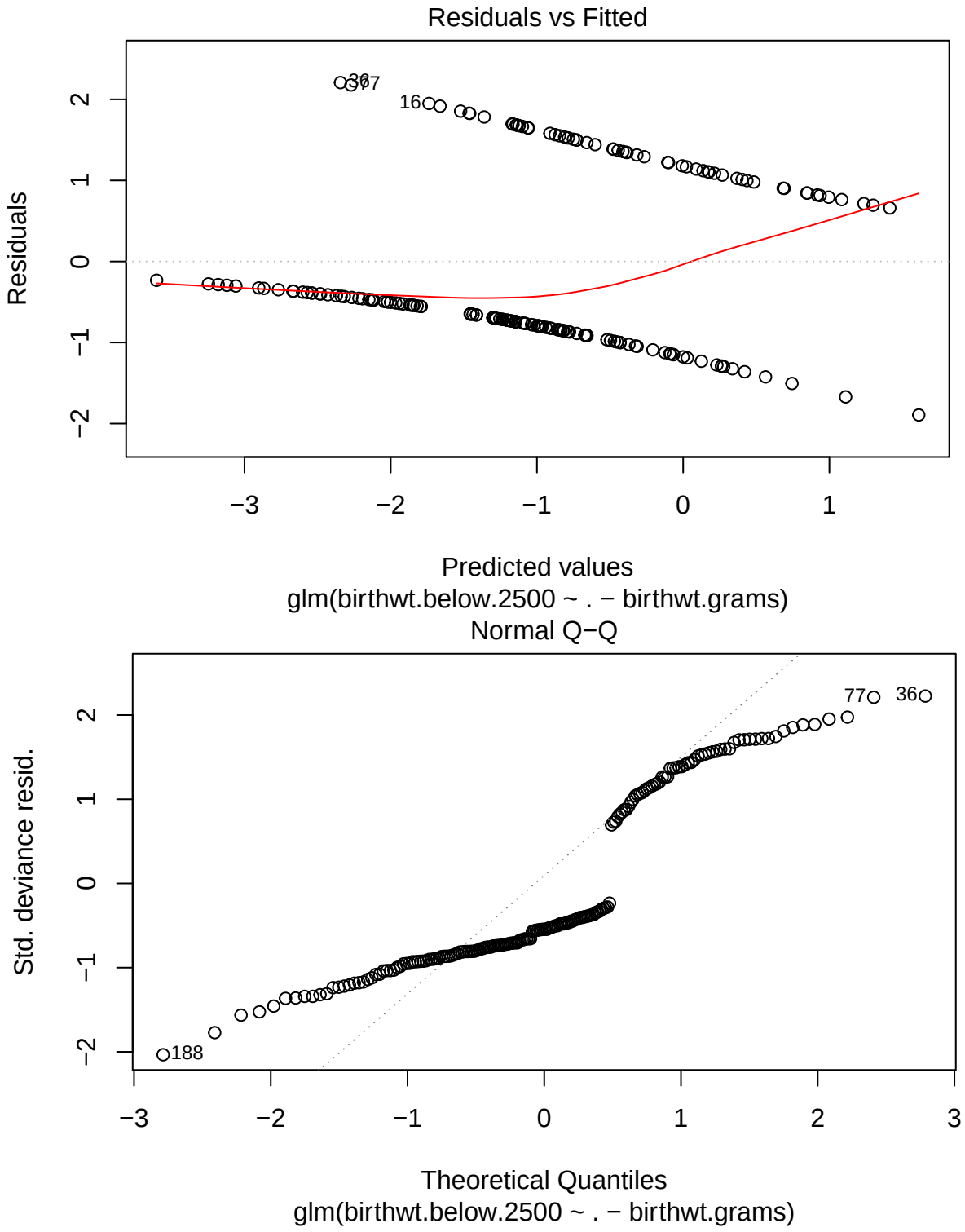
```
glm.1 <- glm(birthwt.below.2500 ~ . - birthwt.grams, data = birthwt.noout, family = binomial(link = log
```

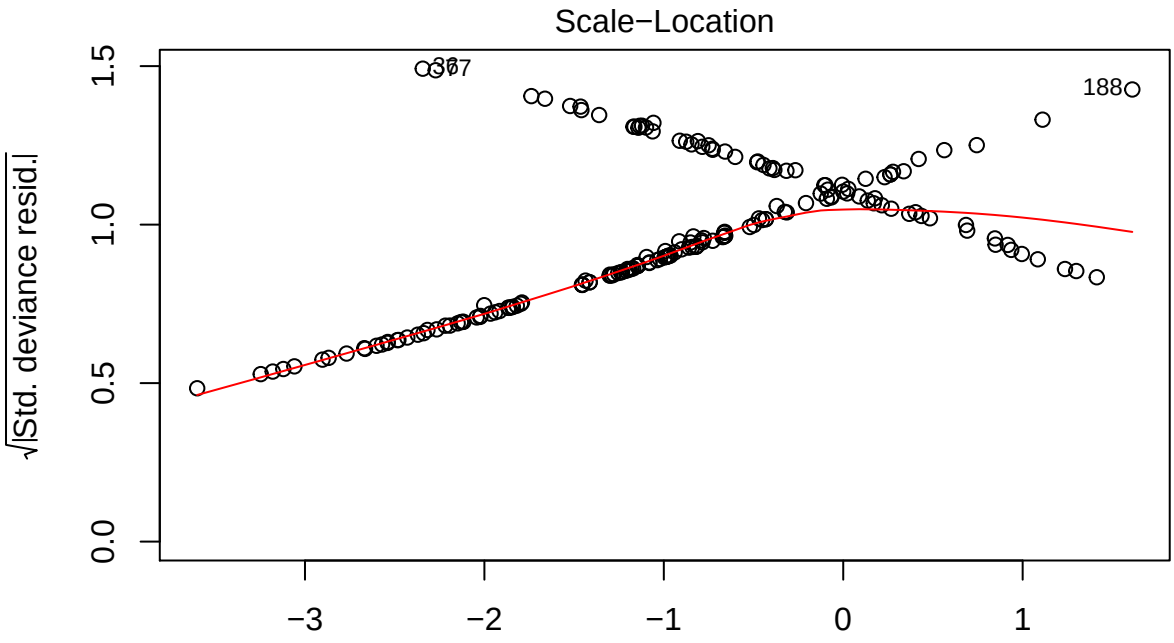
```
summary(glm.1)
```

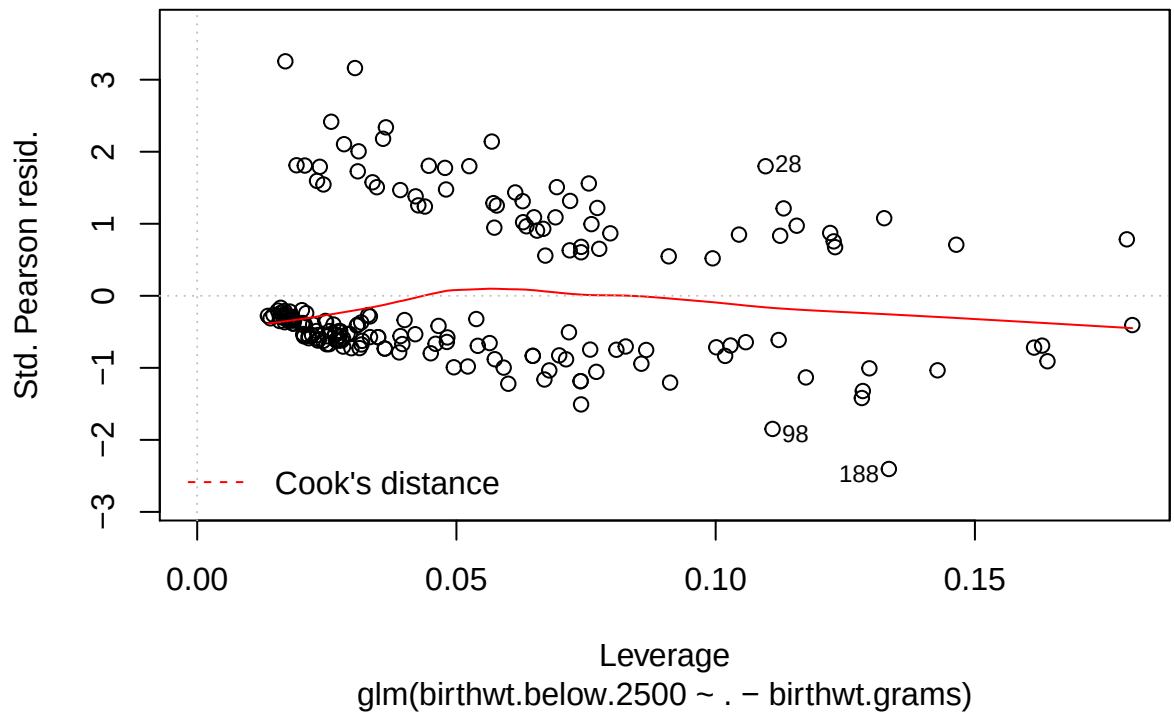
```
##
```

```
## Call:
## glm(formula = birthwt.below.2500 ~ . - birthwt.grams, family = binomial(link = logit),
##     data = birthwt.noout)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8938  -0.8222  -0.5363   0.9848   2.2069
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)          1.721830   1.258897   1.368  0.17140
## mother.age          -0.027537   0.037718  -0.730  0.46534
## mother.weight       -0.015474   0.006919  -2.237  0.02532 *
## raceother           -0.395505   0.537685  -0.736  0.46199
## racewhite           -1.269006   0.527180  -2.407  0.01608 *
## mother.smokesYes     0.931733   0.402359   2.316  0.02058 *
## previous.prem.labor  0.539549   0.345413   1.562  0.11828
## hypertensionYes      1.860521   0.697502   2.667  0.00764 **
## uterine.irrYes       0.766517   0.458951   1.670  0.09489 .
## physician.visits     0.063402   0.172431   0.368  0.71310
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 233.92  on 187  degrees of freedom
## Residual deviance: 201.15  on 178  degrees of freedom
## AIC: 221.15
##
## Number of Fisher Scoring iterations: 4
```

```
plot(glm.1)
```


Residuals vs Fitted
Residuals
Predicted values
glm(birthwt.below.2500 ~ . − birthwt.grams)
Normal Q−Q
Std. deviance resid.
Theoretical Quantiles
glm(birthwt.below.2500 ~ . − birthwt.grams)
77
36
188





Why?

Let's take a subset of this data to do predictions.

```r
odds <- seq(1, nrow(birthwt.noout), by = 2)
birthwt.in <- birthwt.noout[odds, ]
birthwt.out <- birthwt.noout[-odds, ]
linear.model.half <- lm(birthwt.grams ~ . - birthwt.below.2500, data = birthwt.in)
```

```r
summary(linear.model.half)
```

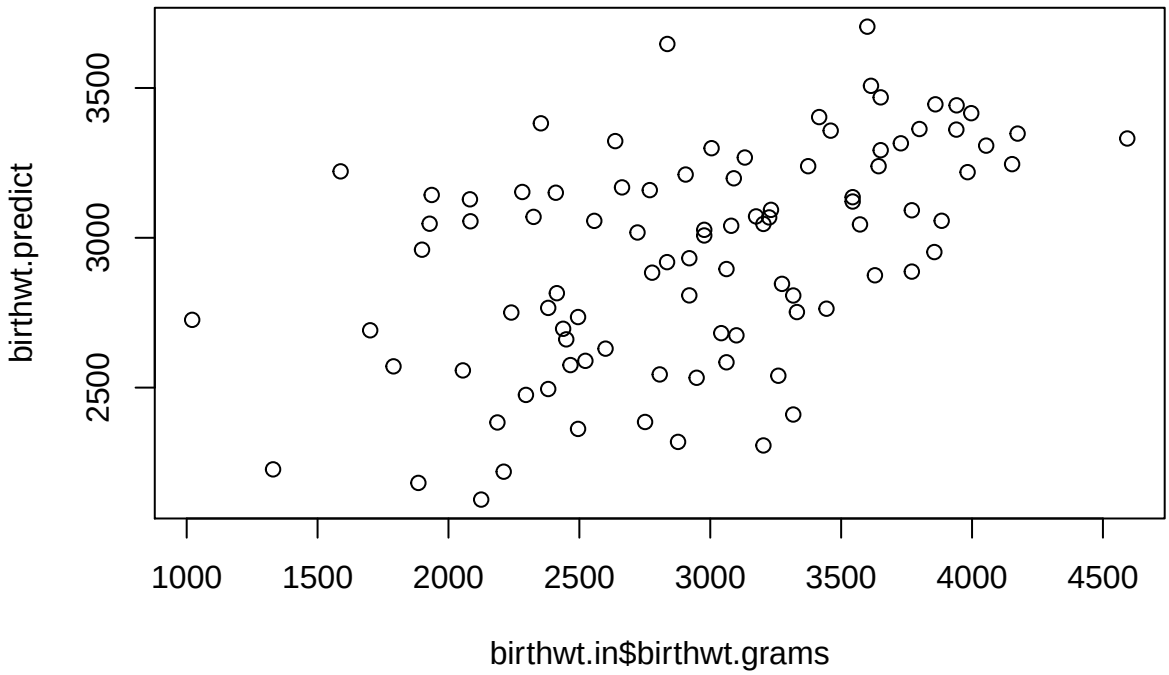
```
##
## Call:
## lm(formula = birthwt.grams ~ . - birthwt.below.2500, data = birthwt.in)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1705.17  -303.11    26.48   427.18  1261.57
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)         2514.891    450.245   5.586 2.81e-07 ***
## mother.age             7.052     14.935   0.472  0.63801
## mother.weight          2.683      2.885   0.930  0.35501
## raceother            113.948    224.519   0.508  0.61312
## racewhite            466.219    204.967   2.275  0.02548 *
## mother.smokesYes    -217.218    154.521  -1.406  0.16349
## previous.prem.labor -206.093    143.726  -1.434  0.15530
## hypertensionYes     -653.594    281.795  -2.319  0.02280 *
## uterine.irrYes      -547.884    193.386  -2.833  0.00577 **
## physician.visits    -130.202     81.400  -1.600  0.11346
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 643.7 on 84 degrees of freedom
## Multiple R-squared:  0.2585, Adjusted R-squared:  0.1791
## F-statistic: 3.254 on 9 and 84 DF,  p-value: 0.001942
```

Prediction of Training Data

```r
birthwt.predict <- predict(linear.model.half)
cor(birthwt.in$birthwt.grams, birthwt.predict)
```

```
## [1] 0.508442
```

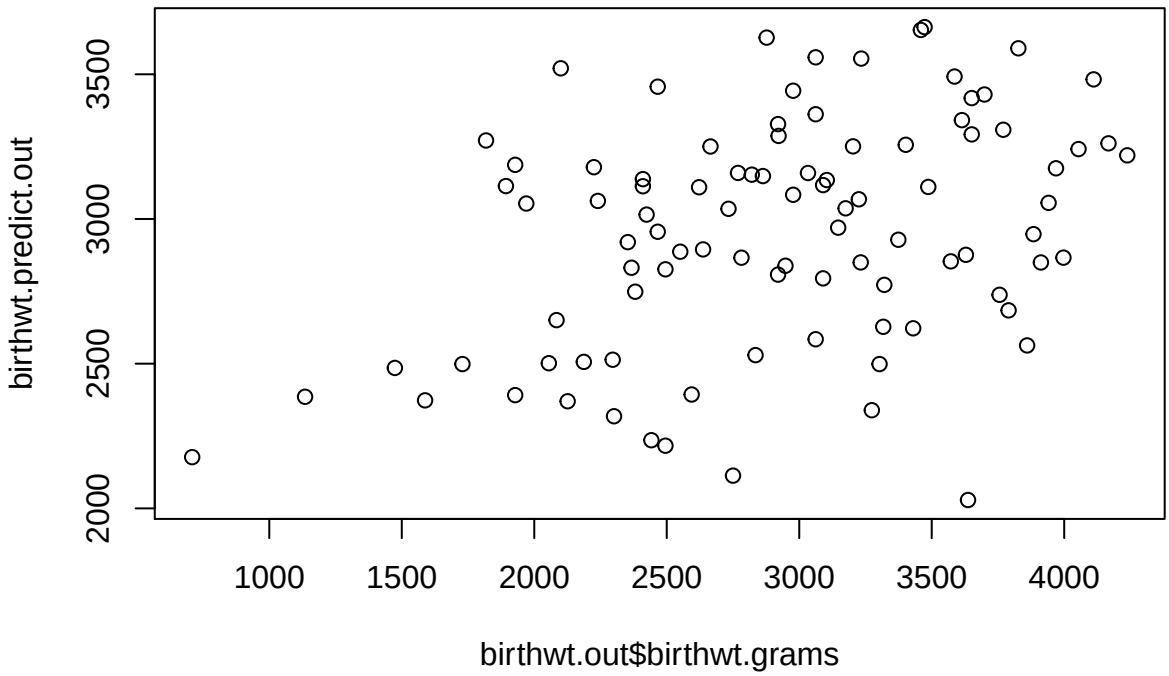
```r
plot(birthwt.in$birthwt.grams, birthwt.predict)
```




```
birthwt.predict.out <- predict(linear.model.half, birthwt.out)
cor(birthwt.out$birthwt.grams, birthwt.predict.out)
```

```
## [1] 0.3749431
```

```
plot(birthwt.out$birthwt.grams, birthwt.predict.out)
```



Summary

- Loading and saving R objects is very easy

- Reading and writing dataframes is pretty easy
- Linear models are very easy via `lm()`
- Generalized linear models are pretty easy via `glm()`
- Generalized linear mixed models via `lme4()` and `glmm()`

Lecture 7: Distributions

Agenda

- Random number generation
- Distributions in R
- Distributional Models
- Moments, generalized moments, likelihood
- Visual comparisons and basic testing

Random number generation

How does R get "random" numbers?

- It doesn't, instead it uses pseudorandom numbers that we hope are indistinguishable from random numbers
- Pseudorandom generators produce a deterministic sequence that is indistinguishable from a true random sequence if you don't know how it started
- Why? Truly random numbers are expensive
- Pseudorandom generators produce a sequence of Uniform$(0, 1)$ random variates
- Linear congruential generator is one of the oldest and best-known pseudorandom number generator algorithms
- Other distributions usually based such a sequence
- Example: If $U \sim \text{Unif}(0, 1)$ then $-\beta \ln(1 - U) \sim \text{Exp}(\beta)$
- More on this later ...

Example: `runif()`

```
runif(1:10)
```

```
##  [1] 0.36254310 0.62119425 0.20701712 0.82909934 0.03653893 0.53780104
##  [7] 0.42990546 0.64734855 0.60896039 0.25099778
```

```
set.seed(10)
runif(1:10)
```

```
##  [1] 0.50747820 0.30676851 0.42690767 0.69310208 0.08513597 0.22543662
##  [7] 0.27453052 0.27230507 0.61582931 0.42967153
```

```
set.seed(10)
runif(1:10)
```

```
##  [1] 0.50747820 0.30676851 0.42690767 0.69310208 0.08513597 0.22543662
##  [7] 0.27453052 0.27230507 0.61582931 0.42967153
```

Linear congruential generator

- Easily implemented and fast
- Modulo arithmetic via storage-bit truncation
- Defined as

$$X_{n+1} = (aX_n + c) \mod m$$

where X is the sequence of pseudorandom values

```
seed <- 10
new.random <- function(a = 5, c = 12, m = 16) {
    out <- (a * seed + c)%%m
    seed <<- out
    return(out)
}
out.length <- 20
variates <- rep(NA, out.length)
for (kk in 1:out.length) variates[kk] <- new.random()
variates
```

```
##  [1] 14  2  6 10 14  2  6 10 14  2  6 10 14  2  6 10 14  2  6 10
```

Unfortunately, this sequence has period 8

```
variates <- rep(NA, out.length)
for (kk in 1:out.length) variates[kk] <- new.random(a = 131, c = 7, m = 16)
variates
```

```
##  [1]  5  6  9  2 13 14  1 10  5  6  9  2 13 14  1 10  5  6  9  2
```

```
variates <- rep(NA, out.length)
for (kk in 1:out.length) variates[kk] <- new.random(a = 129, c = 7, m = 16)
variates
```

```
##  [1]  9  0  7 14  5 12  3 10  1  8 15  6 13  4 11  2  9  0  7 14
```

```
variates <- rep(NA, out.length)
for (kk in 1:out.length) variates[kk] <- new.random(a = 1664545, c = 1013904223,
    m = 2^32)
variates
```

```
##  [1] 1037207853 2090831916 4106096907  768378826 3835752553 1329121000
##  [7] 2125006663 2668506502 3581687205 2079234980 2067291011 2197025090
## [13] 3748878561 2913996384  758844863 4029469438 2836748829 1458315036
## [19] 2399149563 2766656186
```

How to distinguish non-randomness?

- Look at period
- Missing some values
- Proper distribution in the limit
- Autocorrelation
- Gets harder for higher dimensions

Distributions

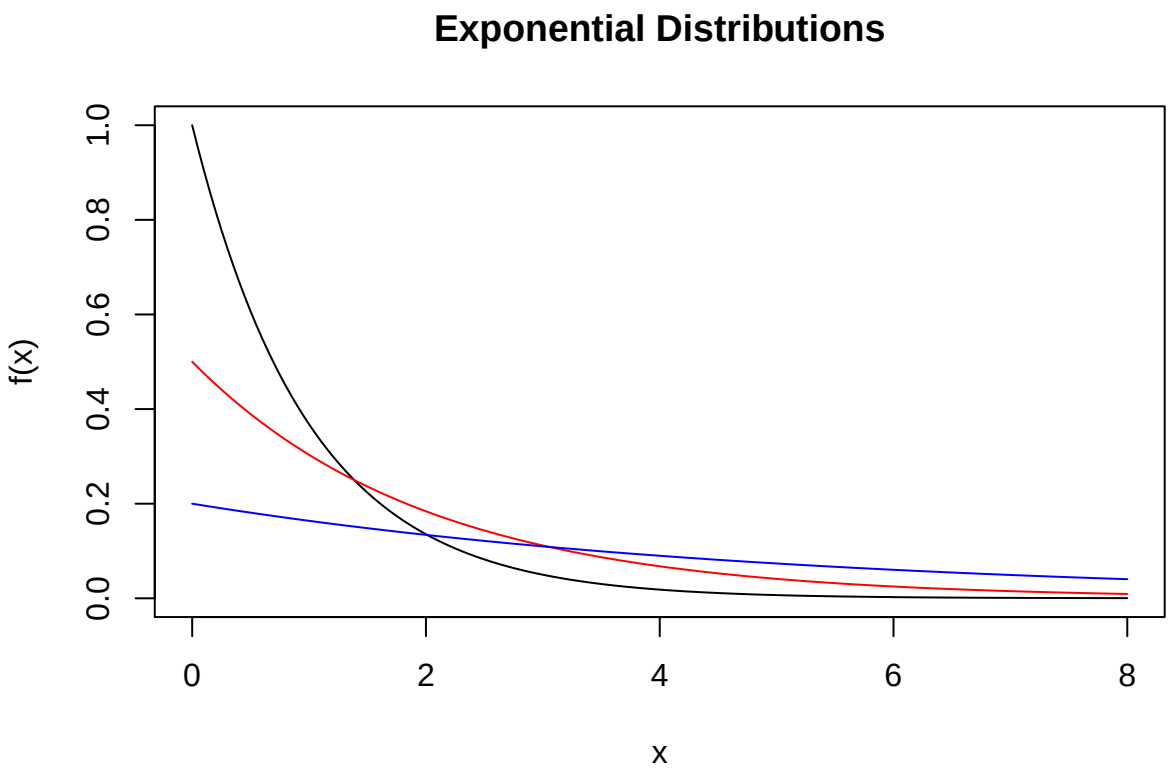
- Beta, Binomial, Cauchy, Chi-Square, Exponential, F, Gamma, Geometric, Hypergeometric, Logistic, Log Normal, Negative Binomial, Normal, Poisson, Student t, Studentized Range, Uniform, Weibull, Wilcoxon Rank Sum Statistic, Wilcoxon Signed Rank Statistic
- Parameters of these distributions may not agree with textbooks
- Lots of distributions, but they all work the same way

Every distribution that R handles has four functions. There is a root name, for example, the root name for the normal distribution is `norm`. This root is prefixed by one of the letters

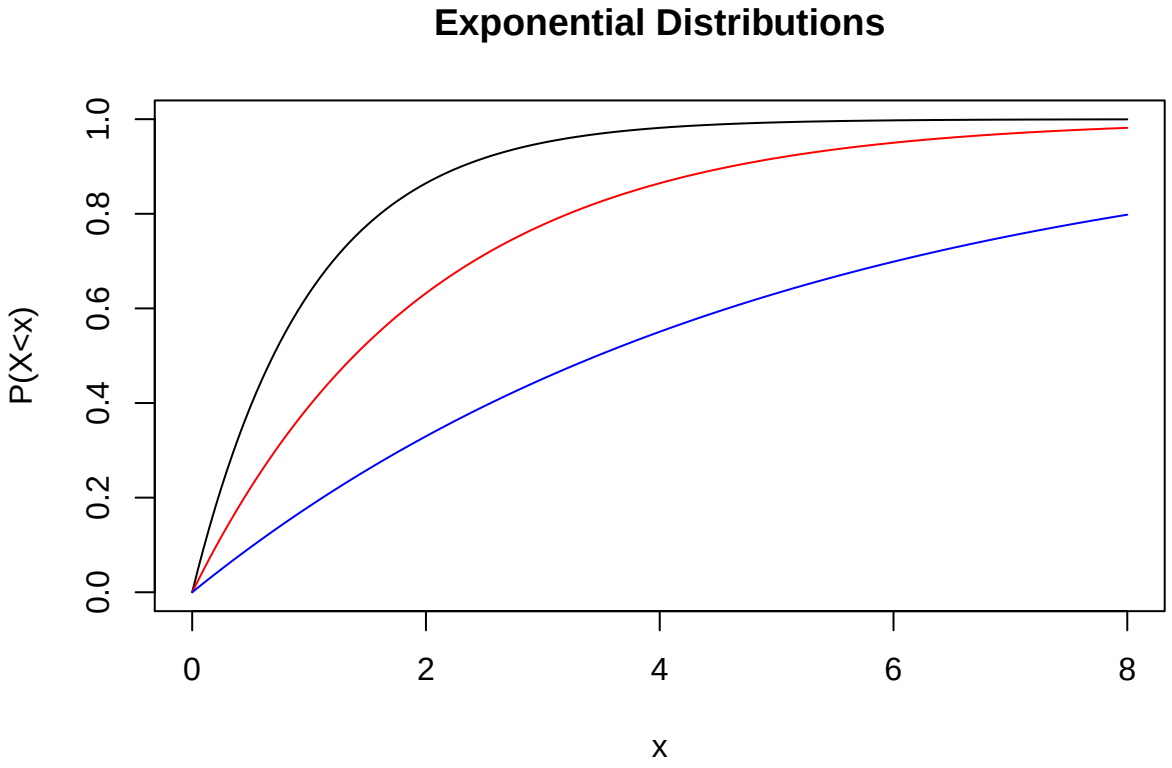
- `p` for "probability", the cumulative distribution function (c. d. f.)
- `q` for "quantile", the inverse c. d. f.
- `d` for "density", the density function (p. f. or p. d. f.)
- `r` for "random", a random variable having the specified distribution
- Beta, Binomial, Cauchy, Chi-Square, Exponential, F, Gamma, Geometric, Hypergeometric, Logistic, Log Normal, Negative Binomial, Normal, Poisson, Student t, Studentized Range, Uniform, Weibull, Wilcoxon Rank Sum Statistic, Wilcoxon Signed Rank Statistic
- `beta`, `binom`, `cauchy`, `chisq`, `exp`, `f`, `gamma`, `geom`, `hyper`, `logis`, `lnorm`, `nbinom`, `norm`, `pois`, `t`, `tukey`, `unif`, `weibull`, `wilcox`, `signrank`
- Exponential distribution has `pexp`, `qexp`, `dexp`, and `rexp`

Exponential distribution

```
this.range <- seq(0, 8, 0.05)
plot(this.range, dexp(this.range), ty = "l", main = "Exponential Distributions",
    xlab = "x", ylab = "f(x)")
lines(this.range, dexp(this.range, rate = 0.5), col = "red")
lines(this.range, dexp(this.range, rate = 0.2), col = "blue")
```




```
this.range <- seq(0, 8, 0.05)
plot(this.range, pexp(this.range), ty = "l", main = "Exponential Distributions",
    xlab = "x", ylab = "P(X<x)")
lines(this.range, pexp(this.range, rate = 0.5), col = "red")
lines(this.range, pexp(this.range, rate = 0.2), col = "blue")
```



Fitting distributional models

- Method of moments
- Match other summary statistics
- Maximize likelihood estimation

Method of moments: Closed form

- Pick enough moments to identify the parameters, i.e. at least 1 moment per parameter
- Write moment equations in terms of the parameters, e.g. the gamma distribution

$$\mu = as \text{ and } \sigma^2 = as^2$$

- Solve the system of equations (by hand)

$$a = \mu^2/\sigma^2 \text{ and } s = \sigma^2/\mu$$

- Write a function

```
gamma.est_MM <- function(x) {
    m <- mean(x)
    v <- var(x)
    return(c(shape = m^2/v, scale = v/m))
}
```

Method of moments: Numerically

- Write functions to get moments from parameters (usually algebra)

- Set up the difference between data and model as another function

```
gamma.mean <- function(shape, scale) {
    return(shape * scale)
}
gamma.var <- function(shape, scale) {
    return(shape * scale^2)
}
gamma.discrepancy <- function(shape, scale, x) {
    return((mean(x) - gamma.mean(shape, scale))^2 + (var(x) - gamma.mean(shape,
        scale))^2)
}
```

- Minimize your function

Applies more generally

- Can match other data summaries, e.g. the median or ratios of quantiles
- If you can't solve exactly, set up a discrepancy function and minimize it

Maximum Likeihood

- Usually think of parameters as fixed and consider the probability of different outcomes, $f(x|\theta)$ with θ constant and x changing
- Likelihood of a parameter value, i.e. $L(\theta)$ treats this a function of θ
- Search over values of θ for the one that makes the data most likely
- Results in maximum likelihood estimate (MLE)
- Suppose $x_1, \ldots, x_n$ are i.i.d., then

$$L(\theta) = \prod_{i=1}^{n} f(x_i|\theta)$$

- Tends to be computationally unstable since it may contain lots of small numbers
- Instead consider the log likelihood

$$l(\theta) = \sum_{i=1}^{n} \log f(x_i|\theta)$$

- In pseudo-code

```
loglike.foo <- function(params, x) {
    sum(dfoo(x = x, params, log = TRUE))
}
```

What to do with the likelihood?

- Maximize it!
- Sometimes we can do the maximization by hand via calculus
- Gaussian, Poisson, Pareto, ...
- More generally we'll consider numerical optimization
- Include a minus sign when using a minimization function

Why use an MLE?

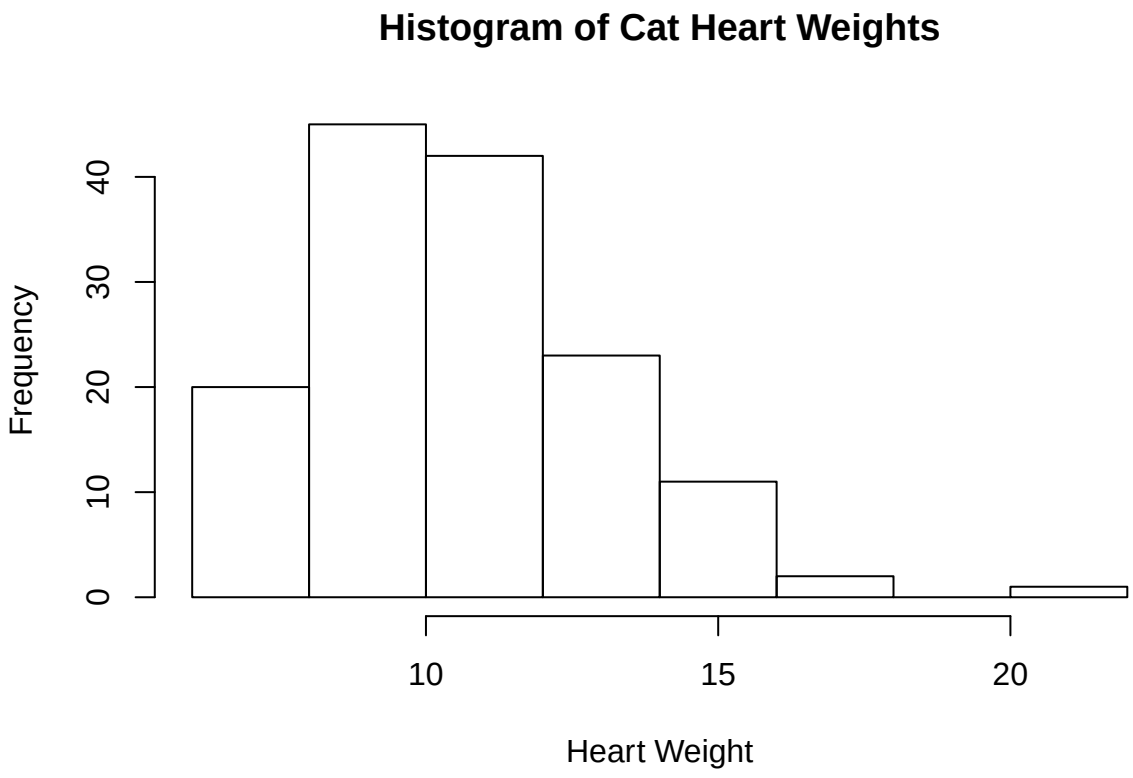
- Usually consistent: converges on the truth as we get more data
- Usually efficient: converges on the truth at least as fast as anything else
- Some settings where the maximum isn't well-defined (e.g. x_{min} for a Pareto)
- Sometimes data is too aggregated or mangled to use an MLE

MLEs for univariate distributions

- `fitdistr` in `MASS` package
- Knows most standard distributions, but can also handle arbitrary probability density functions
- Starting value for the optimization is optional for some distributions, required for others (including user-defined densities)
- Returns parameter estimates and standard errors from large-n approximations (so use cautiously)

Example: Cat heart weights

```
library(MASS)
data("cats", package = "MASS")
hist(cats$Hwt, xlab = "Heart Weight", main = "Histogram of Cat Heart Weights")
```

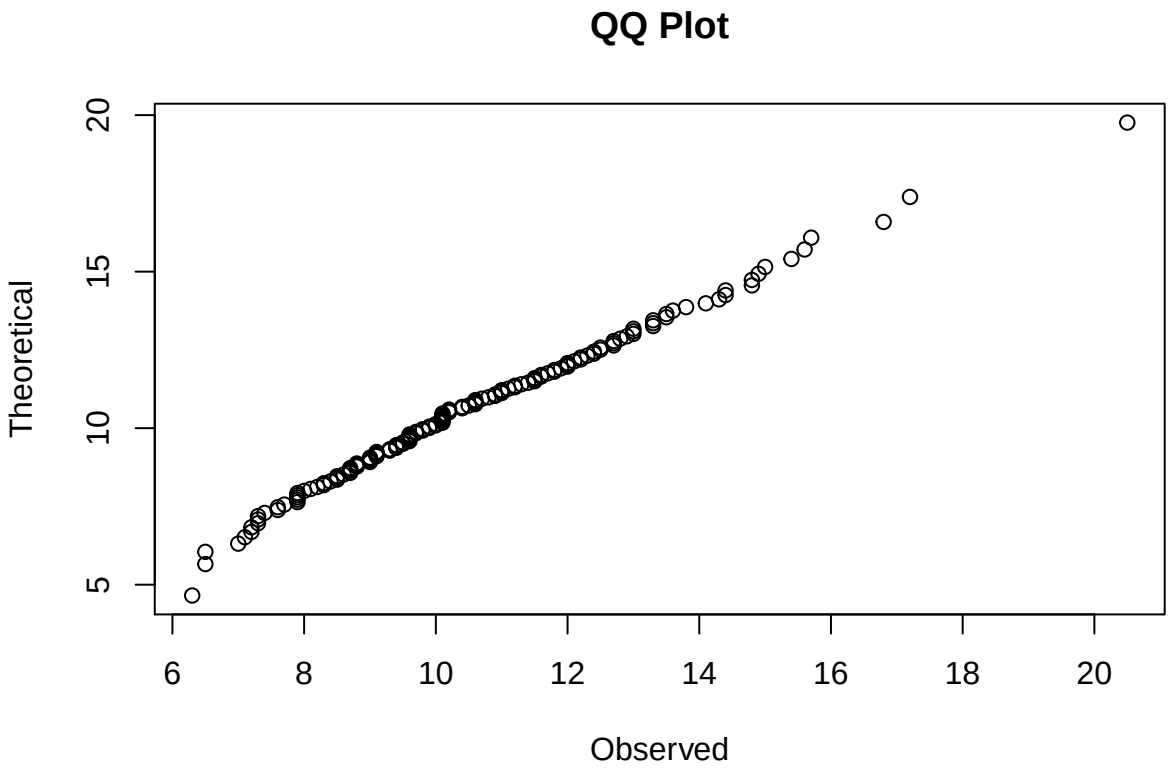



```
fitdistr(cats$Hwt, densfun = "gamma")
```

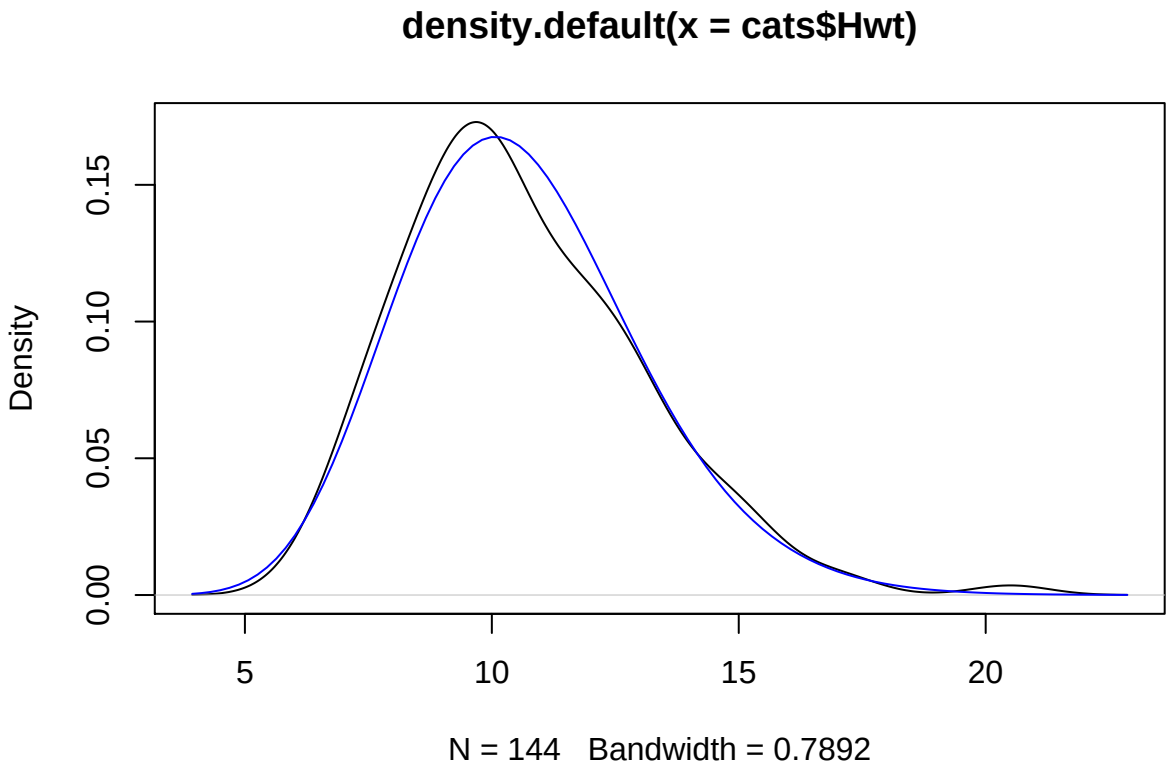
```
##      shape         rate
##   20.2998092    1.9095724
##  ( 2.3729250) ( 0.2259942)
```

Checking the fit

```
cats.gamma <- gamma.est_MM(cats$Hwt)
qqplot(cats$Hwt, qgamma(ppoints(500), shape = cats.gamma["shape"], scale = cats.gamma["scale"]),
    xlab = "Observed", ylab = "Theoretical", main = "QQ Plot")
```



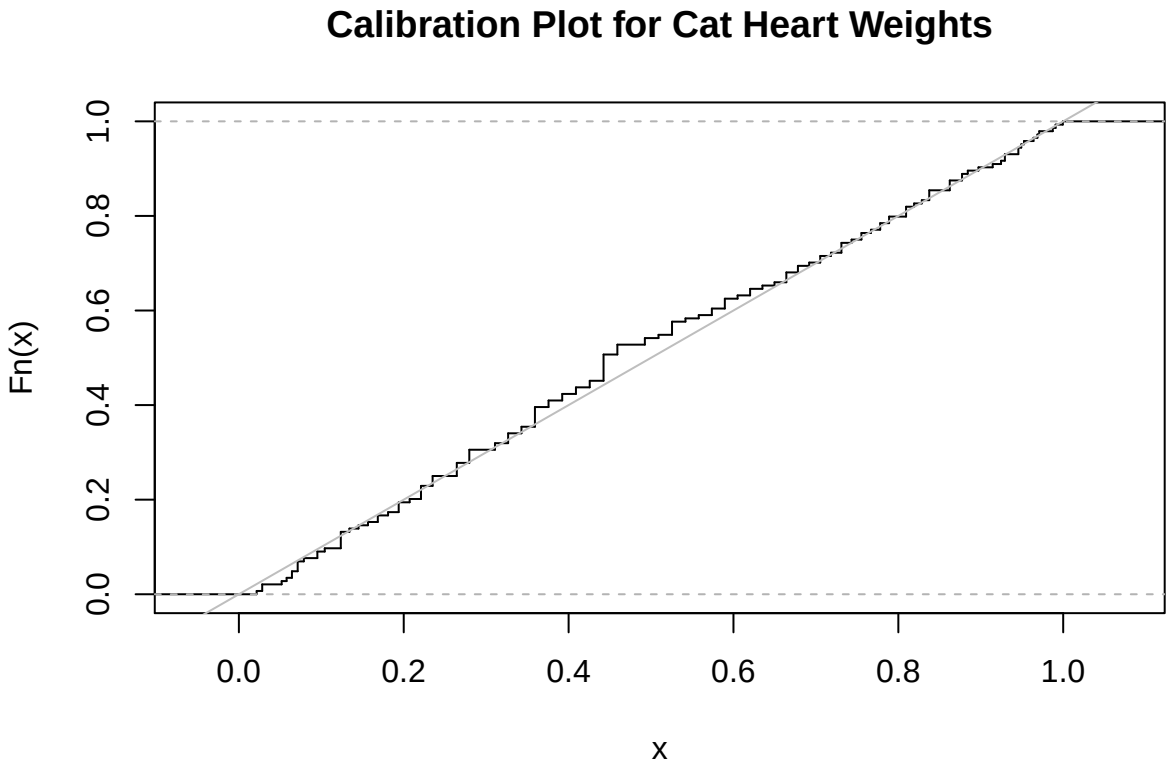

```
plot(density(cats$Hwt))  # more on this later
curve(dgamma(x, shape = cats.gamma["shape"], scale = cats.gamma["scale"]), add = TRUE,
    col = "blue")
```



Calibration plots

- If the distribution is right, 50% of the data should be below the median, 90% should be below the 90th percentile, etc.
- Special case of calibration of probabilities: events with probability p% should happen about p% of the time, not more and not less
- Can look at calibration by calculating the (empirical) CDF of the (theoretical) CDF and plotting
- Ideal calibration plot is a straight line up the diagonal
- Systematic deviations are a warning sign

```
plot(ecdf(pgamma(cats$Hwt, shape = cats.gamma["shape"], scale = cats.gamma["scale"])),
    main = "Calibration Plot for Cat Heart Weights", verticals = T, pch = "")
abline(0, 1, col = "grey")
```



Exercise: Write a general function making a calibration plot that inputs a data vector, cumulative probability function, and parameter vector

Kolmogorov-Smirnov test

- How much "error" is acceptable in a QQ plot or calibration plot?
- Alternatively, consider the biggest gap between theoretical and empirical CDF:

$$D_{KS} = \max_x \left| F(x) - \hat{F}(x) \right|$$

- Useful because D_{KS} has the same distribution if the theoretical CDF is fixed and correct
- Also works for comparing the empirical CDFs of two samples, to see if they came from the same distribution

```
ks.test(cats$Hwt, pgamma, shape = cats.gamma["shape"], scale = cats.gamma["scale"])
```

```
## Warning in ks.test(cats$Hwt, pgamma, shape = cats.gamma["shape"], scale =
## cats.gamma["scale"]): ties should not be present for the Kolmogorov-Smirnov
## test
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  cats$Hwt
## D = 0.068637, p-value = 0.5062
## alternative hypothesis: two-sided
```

- Caution: Solution is more complicated (and not properly handled by built-in R) if parameters are estimated
- Estimating parameters makes the fit look better than it really is
- Could estimate using (say) 80% of the data, and then check the fit on the remaining 20%

- Can also test whether two samples come from same distribution

```
x <- rnorm(100, mean = 0, sd = 1)
y <- rnorm(100, mean = 0.5, sd = 1)
ks.test(x, y)
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  x and y
## D = 0.28, p-value = 0.0007873
## alternative hypothesis: two-sided
```

Chi-squared test

- Compare an actual table of counts to a hypothesized probability distribution

```
M <- as.table(rbind(c(762, 327, 468), c(484, 239, 477)))
dimnames(M) <- list(gender = c("F", "M"), party = c("Democrat", "Independent",
    "Republican"))
M
```

```
##       party
## gender Democrat Independent Republican
##      F      762         327        468
##      M      484         239        477
```

```
Xsq <- chisq.test(M)
Xsq
```

```
##
##  Pearson's Chi-squared test
##
## data:  M
## X-squared = 30.07, df = 2, p-value = 2.954e-07
```

- The p-value calculated by `chisq.test()` assumes that all the probabilities in `p` were fixed, not estimated from the data used for testing, so df = number of cells in the table â^'1
- If we estimate `q` parameters, we need to subtract `q` degrees of freedom

Chi-squared test for continuous distributions

- Divide the range into bins and count the number of observations in each bin; this will be `x` in `chisq.test()`
- Use the CDF function `p foo` to calculate the theoretical probability of each bin; this is `p`
- Plug in to `chisq.test`
- If parameters are estimated, adjust

`hist()` gives us break points and counts:

```
cats.hist <- hist(cats$Hwt, plot = FALSE)
cats.hist$breaks
```

```
## [1]  6  8 10 12 14 16 18 20 22
```

```
cats.hist$counts
```

```
## [1] 20 45 42 23 11  2  0  1
```

Use these for a χ^2 test

```
# Why the padding by -Inf and Inf?
p <- diff(pgamma(c(-Inf, cats.hist$breaks, Inf), shape = cats.gamma["shape"],
    scale = cats.gamma["scale"]))
# Why the padding by 0 and 0?
x2 <- chisq.test(c(0, cats.hist$counts, 0), p = p)$statistic
```

```
## Warning in chisq.test(c(0, cats.hist$counts, 0), p = p): Chi-squared
## approximation may be incorrect
```

```
# Why +2? Why -length(cats.gamma)?
pchisq(x2, df = length(cats.hist$counts) + 2 - length(cats.gamma))
```

```
## X-squared
##  0.854616
```

Don't need to run `hist` first; can also use `cut` to discretize

There are better ways to do this

- Loss of information from discretization
- Lots of work just to use `chisq.test()`
- Try `ks.test`, "bootstrap" testing, "smooth tests of goodness of fit", ...

Summary

- Random number generation
- Distributions in R
- Parametric distributions are models
- Methods of fitting; moments, generalized moments, likelihood
- Methods of checking; visual comparisons, other statistics, tests, calibration

Lecture 8: Optimization I

Agenda

- Functions are objects: can be arguments for or returned by other functions
- Example: `curve()`
- Optimization via gradient descent, Newton's method, Nelder-Mead, ...
- Curve-fitting by optimizing

Functions as Objects

- In R, functions are objects, just like everything else
- This means that they can be passed to functions as arguments and returned by functions as outputs as well
- We often want to do very similar things to many different functions
- The procedure is the same, only the function we're working with changes
- Write one function to do the job, and pass the function as an argument
- Because R treats a function like any other object, we can do this simply: invoke the function by its argument name in the body
- We have already seen examples
- `apply()`, `sapply()`, etc.: Take *this* function and use it on all of *these* objects
- `nlm()`: Take *this* function and try to make it small, starting from *here*
- `ks.test()`: Compare *these* data to *this* cumulative distribution function
- `curve()`: Evaluate *this* function over *that* range, and plot the results

Syntax Facts About Functions

- Typing a function's name, without parentheses, in the terminal gives you its source code
- Functions are their own **class** in R

```
class(sin)
```

```
## [1] "function"
```

```
class(sample)
```

```
## [1] "function"
```

```
resample <- function(x) {
    sample(x, size = length(x), replace = TRUE)
}
class(resample)
```

```
## [1] "function"
```

- Functions can be put into lists or even arrays
- A call to `function` returns a function object
- body executed: access with `body(foo)`
- arguments required: access with `formals(foo)`
- parent environment: access with `environment(foo)`

R has separate **types** for built-in functions and for those written in R:

```
typeof(resample)
```

```
## [1] "closure"
```

```
typeof(sample)
```

```
## [1] "closure"
```

```
typeof(sin)
```

```
## [1] "builtin"
```

Why `closure` for written-in-R functions? Because expressions are "closed" by referring to the parent environment

There's also a 2nd class of built-in functions called `primitive`

Anonymous Functions

- `function()` returns an object of class `function`
- So far we've assigned that object to a name
- If we don't have an assignment, we get an **anonymous function**
- Usually part of some larger expression:

```
sapply((-2):2, function(log.ratio) {
    exp(log.ratio)/(1 + exp(log.ratio))
})
```

```
## [1] 0.1192029 0.2689414 0.5000000 0.7310586 0.8807971
```

- Often handy when connecting other pieces of code
  - especially in things like `apply` and `sapply`
- Won't cluttering the workspace
- Can't be examined or re-used later

Example: grad()

- Many problems in statistics come down to optimization
- So do lots of problems in economics, physics, CS, biology, ...
- Lots of optimization problems require the gradient of the **objective function**
- Gradient of f at x:

$$\nabla f(x) = \left[\left.\frac{\partial f}{\partial x_1}\right|_x \cdots \left.\frac{\partial f}{\partial x_p}\right|_x \right]$$

- We do the same thing to get the gradient of f at x no matter what f is
- Find the partial derivative of f with respect to each component of x and return the vector of partial derivatives
- It makes no sense to re-write this every time we change f!
- Write code to calculate the gradient of an arbitrary function
- We *could* write our own, but there are lots of tricky issues
  - Best way to calculate partial derivative
  - What if x is at the edge of the domain of f?
- Fortunately, someone has already done this

From the package `numDeriv`

```
grad(func, x, ...)
```

- Assumes `func` is a function which returns a single floating-point value
- Assumes `x` is a vector of arguments to `func`
  - If `x` is a vector and `func(x)` is also a vector, then it's assumed `func` is vectorized and we get a vector of derivatives
- Extra arguments in `...` get passed along to `func`
- Other functions in the package for the Jacobian of a vector-valued function, and the matrix of 2nd partials (Hessian)

```
require("numDeriv")
```

```
## Loading required package: numDeriv
```

```
just_a_phase <- runif(n = 1, min = -pi, max = pi)
all.equal(grad(func = cos, x = just_a_phase), -sin(just_a_phase))
```

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

```
phases <- runif(n = 10, min = -pi, max = pi)
all.equal(grad(func = cos, x = phases), -sin(phases))
```

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

```
grad(func = function(x) {
    x[1]^2 + x[2]^3
}, x = c(1, -1))
```

```
## [1] 2 3
```

Note: `grad` is perfectly happy with `func` being an anonymous function!

gradient.descent()

Now we can use this as a piece of a larger machine:

```
gradient.descent <- function(f,x,max.iterations,step.scale,
  stopping.deriv,...) {
  for (iteration in 1:max.iterations) {
    gradient <- grad(f,x,...)
    if(all(abs(gradient) < stopping.deriv)) { break() }
    x <- x - step.scale*gradient
  }
  fit <- list(argmin=x,final.gradient=gradient,final.value=f(x,...),
    iterations=iteration)
  return(fit)
}
```

Works equally well whether `f` is mean squared error of a regression, ψ error of a regression, (negative log) likelihood, cost of a production plan, ...

Cautions

- *Scoping*: `f` takes values for all names which aren't its arguments from the environment where it was defined, not the one where it is called (e.g., not from inside `grad` or `gradient.descent`)

- *Debugging*: If `f` and `g` are both complicated, avoid debugging `g(f)` as a block; divide the work by writing *very simple* `f.dummy` to debug/test `g`, and debug/test the real `f` separately

Returning Functions

- Functions can be return values like anything else
- Create a linear predictor, based on sample values of two variables

```
make.linear.predictor <- function(x, y) {
    linear.fit <- lm(y ~ x)
    predictor <- function(x) {
        return(predict(object = linear.fit, newdata = data.frame(x = x)))
    }
    return(predictor)
}
```

- The predictor function persists and works, even when the data we used to create it is gone

```
library(MASS)
data(cats)
vet_predictor <- make.linear.predictor(x = cats$Bwt, y = cats$Hwt)
rm(cats)  # Data set goes away
vet_predictor(3.5)  # My cat's body mass in kilograms
```

```
##        1
## 13.76256
```

Example: curve()

- A call to `curve` looks like this:

```
curve(expr, from = a, to = b, ...)
```

- `expr` is some expression involving a variable called `x` which is swept `from` the value `a` `to` the value `b`
- `...` are other plot-control arguments
- `curve` feeds the expression a vector `x` and expects a numeric vector back (so this is fine)

  ```
  curve(x^2 * sin(x))
  ```

We can use our own function in `curve`:

```
psi <- function(x, c = 1) {
    ifelse(abs(x) > c, 2 * c * abs(x) - c^2, x^2)
}
curve(psi(x, c = 10), from = -20, to = 20)
```

Or this

```
curve(psi(x = 10, c = x), from = -20, to = 20)
```

If our function doesn't take vectors to vectors, `curve` becomes unhappy

```
mse <- function(y0, a, Y = gmp$pcgmp, N = gmp$pop) {
    mean((Y - y0 * (N^a))^2)
}
```

```
> curve(mse(a=x,y0=6611),from=0.10,to=0.15)
Error in curve(mse(a = x, y0 = 6611), from = 0.1, to = 0.15) :
  'expr' did not evaluate to an object of length 'n'
In addition: Warning message:
In N^a : longer object length is not a multiple of shorter object length
```

How do we solve this?

Define a new, vectorized function, say with `sapply`:

```
sapply(seq(from = 0.1, to = 0.15, by = 0.01), mse, y0 = 6611)
```

```
## [1] 154701953 102322974  68755654  64529166 104079527 207057513
```

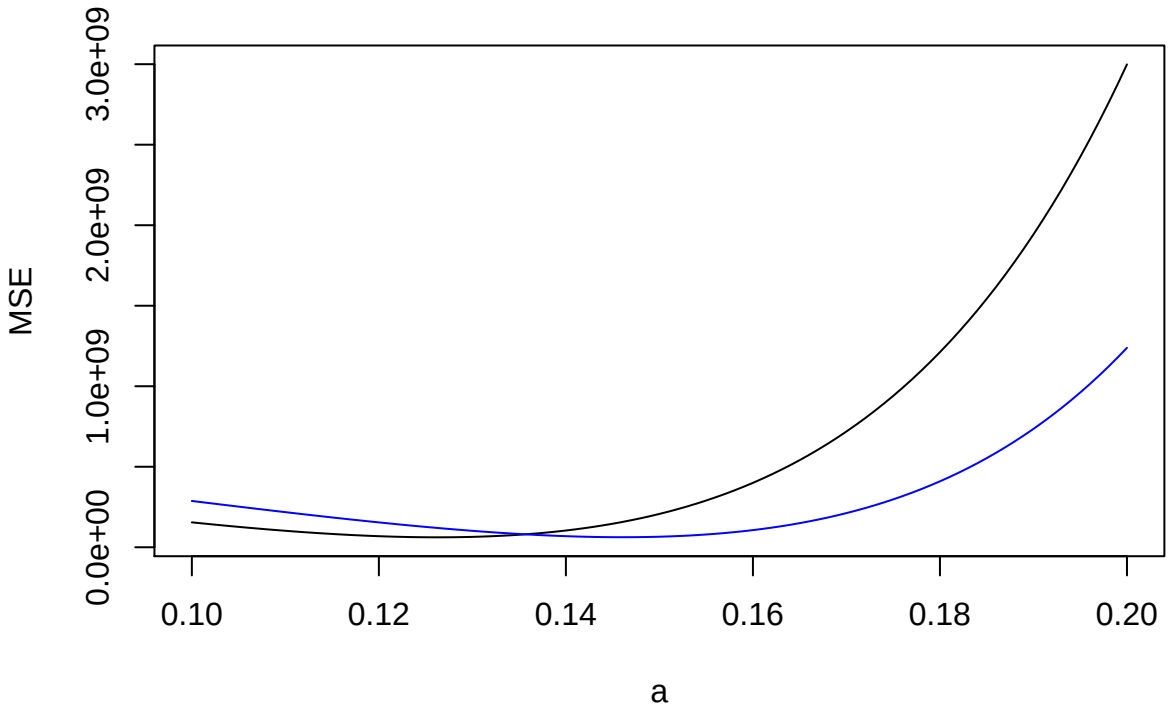
```
mse(6611, 0.1)
```

```
## [1] 154701953
```

```
mse.plottable <- function(a, ...) {
    return(sapply(a, mse, ...))
}
mse.plottable(seq(from = 0.1, to = 0.15, by = 0.01), y0 = 6611)
```

```
## [1] 154701953 102322974  68755654  64529166 104079527 207057513
```

```
curve(mse.plottable(a = x, y0 = 6611), from = 0.1, to = 0.2, xlab = "a", ylab = "MSE")
curve(mse.plottable(a = x, y0 = 5100), add = TRUE, col = "blue")
```



Alternate strategy: `Vectorize()` returns a new, vectorized function

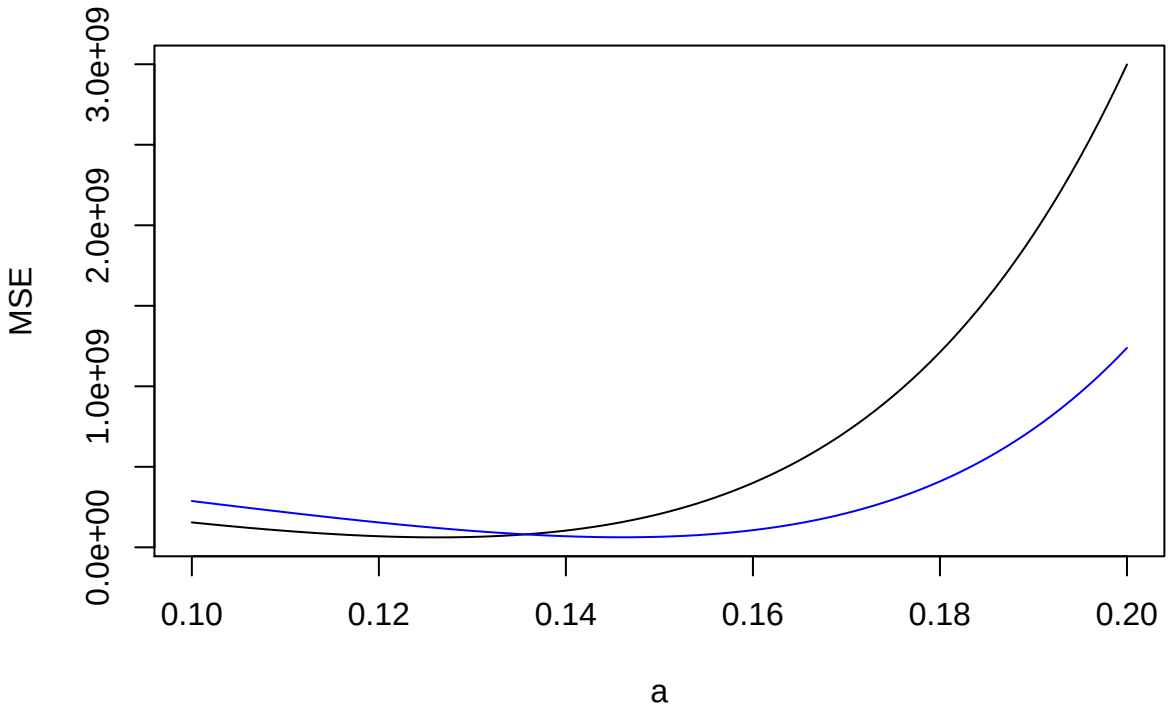
```
mse.vec <- Vectorize(mse, vectorize.args = c("y0", "a"))
mse.vec(a = seq(from = 0.1, to = 0.15, by = 0.01), y0 = 6611)
```

```
## [1] 154701953 102322974  68755654  64529166 104079527 207057513
```

```r
mse.vec(a = 1/8, y0 = c(5000, 6000, 7000))
```

```
## [1] 134617132  74693733  63732256
```

```r
curve(mse.vec(a = x, y0 = 6611), from = 0.1, to = 0.2, xlab = "a", ylab = "MSE")
curve(mse.vec(a = x, y0 = 5100), add = TRUE, col = "blue")
```



Examples of Optimization Problems

- Minimize mean-squared error of regression surface
- Maximize likelihood of distribution
- Maximize output of tanks from given supplies and factories
- Maximize return of portfolio for given volatility
- Minimize cost of airline flight schedule
- Maximize reproductive fitness of an organism

Optimization Problems

Given an **objective function** $f : \mathcal{D} \mapsto R$, find

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, f(\theta)$$

Basics: maximizing f is minimizing $-f$:

$$\underset{\theta}{\operatorname{argmax}}\, f(\theta) = \underset{\theta}{\operatorname{argmin}}\, -f(\theta)$$

If h is strictly increasing (e.g., log), then

$$\underset{\theta}{\operatorname{argmin}}\, f(\theta) = \underset{\theta}{\operatorname{argmin}}\, h(f(\theta))$$

Considerations

- Approximation: How close can we get to θ^*, and/or $f(\theta^*)$?
- Time complexity: How many computer steps does that take? Varies with precision of approximation, niceness of f, size of $\mathcal{D}$, size of data, method...
- Most optimization algorithms use **successive approximation**, so distinguish number of iterations from cost of each iteration

Use calculus

Suppose x is one dimensional and f is smooth. If x^* is an **interior** minimum / maximum / extremum point

$$\left.\frac{df}{dx}\right|_{x=x^*} = 0$$

If x^* a minimum,

$$\left.\frac{d^2f}{dx^2}\right|_{x=x^*} > 0$$

This all carries over to multiple dimensions: At an **interior extremum**,

$$\nabla f(\theta^*) = 0$$

At an **interior minimum**,

$$\nabla^2 f(\theta^*) \geq 0$$

meaning for any vector v,

$$v^T \nabla^2 f(\theta^*) v \geq 0$$

$\nabla^2 f =$ the **Hessian**, **H** θ might just be a **local** minimum

Gradient Descent

$$f'(x_0) = \left.\frac{df}{dx}\right|_{x=x_0} = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

$$f(x) \approx f(x_0) + (x - x_0) f'(x_0)$$

Locally, the function looks linear; to minimize a linear function, move down the slope

Multivariate version:

$$f(\theta) \approx f(\theta_0) + (\theta - \theta_0) \cdot \nabla f(\theta_0)$$

$\nabla f(\theta_0)$ points in the direction of fastest ascent at θ_0

1. Start with initial guess for θ, step-size η
2. While ((not too tired) and (making adequate progress))

- Find gradient $\nabla f(\theta)$
- Set $\theta \leftarrow \theta - \eta \nabla f(\theta)$

3. Return final θ as approximate θ^* Variations: adaptively adjust η to make sure of improvement or search along the gradient direction for minimum

Pro:

- Moves in direction of greatest immediate improvement
- If η is small enough, gets to a local minimum eventually, and then stops

Cons:

- "small enough" η can be really, really small
- Slowness or zig-zagging if components of ∇f are very different sizes How much work do we need?

Scaling

Big-O notation:

$$h(x) = O(g(x))$$

means

$$\lim_{x \to \infty} \frac{h(x)}{g(x)} = c$$

for some $c \neq 0$

e.g., $x^2 - 5000x + 123456778 = O(x^2)$

e.g., $e^x/(1 + e^x) = O(1)$

Useful to look at over-all scaling, hiding details Also done when the limit is $x \to 0$

Gradient Descent

Pro:

- For nice f, $f(\theta) \leq f(\theta^*) + \epsilon$ in $O(\epsilon^{-2})$ iterations + For *very* nice f, only $O(\log \epsilon^{-1})$ iterations
- To get $\nabla f(\theta)$, take p derivatives, $\therefore$ each iteration costs $O(p)$

Con:

- Taking derivatives can slow down as data grows — each iteration might really be $O(np)$

Taylor Series

What if we do a quadratic approximation to f?

$$f(x) \approx f(x_0) + (x - x_0)f'(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0)$$

Special cases of general idea of Taylor approximation

Simplifies if x_0 is a minimum since then $f'(x_0) = 0$:

$$f(x) \approx f(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0)$$

Near a minimum, smooth functions look like parabolas

Carries over to the multivariate case:

$$f(\theta) \approx f(\theta_0) + (\theta - \theta_0) \cdot \nabla f(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T \mathbf{H}(\theta_0)(\theta - \theta_0)$$

Minimizing a Quadratic

If we know

$$f(x) = ax^2 + bx + c$$

we minimize exactly:

$$2ax^* + b = 0x^* = \frac{-b}{2a}$$

If

$$f(x) = \frac{1}{2}a(x - x_0)^2 + b(x - x_0) + c$$

then

$$x^* = x_0 - a^{-1}b$$

Newton's Method

Taylor-expand for the value *at the minimum* θ^*

$$f(\theta^*) \approx f(\theta) + (\theta^* - \theta)\nabla f(\theta) + \frac{1}{2}(\theta^* - \theta)^T \mathbf{H}(\theta)(\theta^* - \theta)$$

Take gradient, set to zero, solve for θ^*:

$$0 = \nabla f(\theta) + \mathbf{H}(\theta)(\theta^* - \theta)\theta^* = \theta - (\mathbf{H}(\theta))^{-1}\nabla f(\theta)$$

Works *exactly* if f is quadratic and $\mathbf{H}^{-1}$ exists, etc.

If f isn't quadratic, keep pretending it is until we get close to θ^*, when it will be nearly true

The Algorithm

1. Start with guess for θ
2. While ((not too tired) and (making adequate progress))

- Find gradient $\nabla f(\theta)$ and Hessian $\mathbf{H}(\theta)$
- Set $\theta \leftarrow \theta - \mathbf{H}(\theta)^{-1}\nabla f(\theta)$

3. Return final θ as approximation to θ^*

Like gradient descent, but with inverse Hessian giving the step-size

"This is about how far you can go with that gradient"

Pros:

- Step-sizes chosen adaptively through 2nd derivatives, much harder to get zig-zagging, over-shooting, etc.
- Also guaranteed to need $O(\epsilon^{-2})$ steps to get within ϵ of optimum
- Only $O(\log\log\epsilon^{-1})$ for very nice functions
- Typically many fewer iterations than gradient descent

Cons:

- Hopeless if $\mathbf{H}$ doesn't exist or isn't invertible
- Need to take $O(p^2)$ second derivatives *plus* p first derivatives
- Need to solve $\mathbf{H}\theta_{\text{new}} = \mathbf{H}\theta_{\text{old}} - \nabla f(\theta_{\text{old}})$ for θ_{new}
- Inverting $\mathbf{H}$ is $O(p^3)$, but cleverness gives $O(p^2)$ for solving for θ_{new}

Getting Around the Hessian

Want to use the Hessian to improve convergence, but don't want to have to keep computing the Hessian at each step

Approaches:

- Use knowledge of the system to get some approximation to the Hessian, use that instead of taking derivatives ("Fisher scoring")
- Use only diagonal entries (p unmixed 2nd derivatives)
- Use $\mathbf{H}(\theta)$ at initial guess, hope $\mathbf{H}$ changes *very* slowly with θ
- Re-compute $\mathbf{H}(\theta)$ every k steps, $k > 1$
- Fast, approximate updates to the Hessian at each step
- Lots of other methods!
- Nedler-Mead, a.k.a. "the simplex method", which doesn't need any derivatives

Nelder-Mead

Try to cage θ^* with a **simplex** of $p+1$ points

Order the trial points, $f(\theta_1) \leq f(\theta_2) \ldots \leq f(\theta_{p+1})$

θ_{p+1} is the worst guess — try to improve it

Center of the not-worst $= \theta_0 = \frac{1}{n}\sum_{i=1}^{n} \theta_i$

Try to improve the worst guess θ_{p+1}

1. **Reflection**: Try $\theta_0 - (\theta_{p+1} - \theta_0)$, across the center from θ_{p+1}

- if it's better than θ_p but not than θ_1, replace the old θ_{p+1} with it
- **Expansion**: if the reflected point is the new best, try $\theta_0 - 2(\theta_{p+1} - \theta_0)$; replace the old θ_{p+1} with the better of the reflected and the expanded point

2. **Contraction**: If the reflected point is worse that θ_p, try $\theta_0 + \frac{\theta_{p+1} - \theta_0}{2}$; if the contracted value is better, replace θ_{p+1} with it
3. **Reduction**: If all else fails, $\theta_i \leftarrow \frac{\theta_1 + \theta_i}{2}$
4. Go back to (1) until we stop improving or run out of time

Making Sense of Nedler-Mead

The Moves:

- Reflection: try the opposite of the worst point
- Expansion: if that really helps, try it some more
- Contraction: see if we overshot when trying the opposite
- Reduction: if all else fails, try making each point more like the best point

Pros:

- Each iteration ≤ 4 values of f, plus sorting (and sorting is at most $O(p \log p)$, usually much better)
- No derivatives used, can even work for dis-continuous f

Con:

- Can need *many* more iterations than gradient methods

Coordinate Descent

Gradient descent, Newton's method, simplex, etc., adjust all coordinates of θ at once — gets harder as the number of dimensions p grows

Coordinate descent: never do more than 1D optimization

- Start with initial guess θ
- While ((not too tired) and (making adequate progress))
- For $i \in (1:p)$
  - do 1D optimization over i^{th} coordinate of θ, holding the others fixed
  - Update i^{th} coordinate to this optimal value
- Return final value of θ

Cons:

- Needs a good 1D optimizer
- Can bog down for very tricky functions, especially with lots of interactions among variables

Pros:

- Can be extremely fast and simple

Curve-Fitting by Optimizing

We have data $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$, and possible curves, $r(x; \theta)$ e.g.

- $r(x) = x \cdot \theta$
- $r(x) = \theta_1 x^{\theta_2}$
- $r(x) = \sum_{j=1}^{q} \theta_j b_j(x)$ for fixed "basis" functions b_j

Least-squares curve fitting:

$$\hat{\theta} = \operatorname*{argmin}_{\theta} \frac{1}{n} \sum_{i=1}^{n} (y_i - r(x_i; \theta))^2$$

"Robust" curve fitting:

$$\hat{\theta} = \operatorname*{argmin}_{\theta} \frac{1}{n} \sum_{i=1}^{n} \psi(y_i - r(x_i; \theta))$$

Summary

- In R, functions are objects, and can be arguments to other functions
- Functions can also be returned by other functions
- Optimizaiton
- Trade-offs: complexity of iteration vs. number of iterations vs. precision of approximation
- Gradient descent: less complex iterations, more guarantees, less adaptive
- Newton: more complex iterations, but few of them for good functions, more adaptive, less robust
- Next time pre-built code like `optim` and `nls`

Lecture 9: Optimization II

Agenda

- Optimization with `optim()` and `nls()`
- Optimization under constraints
- Lagrange multipliers
- Penalized optimization
- Statistical uses of penalized optimization

Optimization in R: optim()

`optim(par, fn, gr, method, control, hessian)`

- `fn`: function to be minimized; mandatory
- `par`: initial parameter guess; mandatory
- `gr`: gradient function; only needed for some methods
- `method`: defaults to a gradient-free method ("Nedler-Mead"), could be BFGS (Newton-ish)
- `control`: optional list of control settings
- (maximum iterations, scaling, tolerance for convergence, etc.)
- `hessian`: should the final Hessian be returned? default FALSE

Return contains the location (`$par`) and the value (`$val`) of the optimum, diagnostics, possibly `$hessian`

```
gmp <- read.table("http://faculty.ucr.edu/~jflegal/206/gmp.dat")
gmp$pop <- gmp$gmp/gmp$pcgmp
library(numDeriv)
mse <- function(theta) {
    mean((gmp$pcgmp - theta[1] * gmp$pop^theta[2])^2)
}
grad.mse <- function(theta) {
    grad(func = mse, x = theta)
}
theta0 = c(5000, 0.15)
fit1 <- optim(theta0, mse, grad.mse, method = "BFGS", hessian = TRUE)
```

fit1: Newton-ish BFGS method

```
fit1[1:3]
```

```
## $par
## [1] 6493.2563738    0.1276921
##
## $value
## [1] 61853983
##
## $counts
## function gradient
##       63       11
```

fit1: Newton-ish BFGS method

```
fit1[4:6]
```

```
## $convergence
## [1] 0
```

```
##
## $message
## NULL
##
## $hessian
##              [,1]         [,2]
## [1,] 5.25021e+01      4422070
## [2,] 4.42207e+06 375729087976
```

Optimization in R: nls()

- `optim` is a general-purpose optimizer
- `nlm` is another general-purpose optimizer; nonlinear least squares
- Try them both if one doesn't work

```
nls(formula, data, start, control, [[many other options]])
```

- `formula`: Mathematical expression with response variable, predictor variable(s), and unknown parameter(s)
- `data`: Data frame with variable names matching `formula`
- `start`: Guess at parameters (optional)
- `control`: Like with `optim` (optional)

Returns an `nls` object, with fitted values, prediction methods, etc. The default optimization is a version of Newton's method.

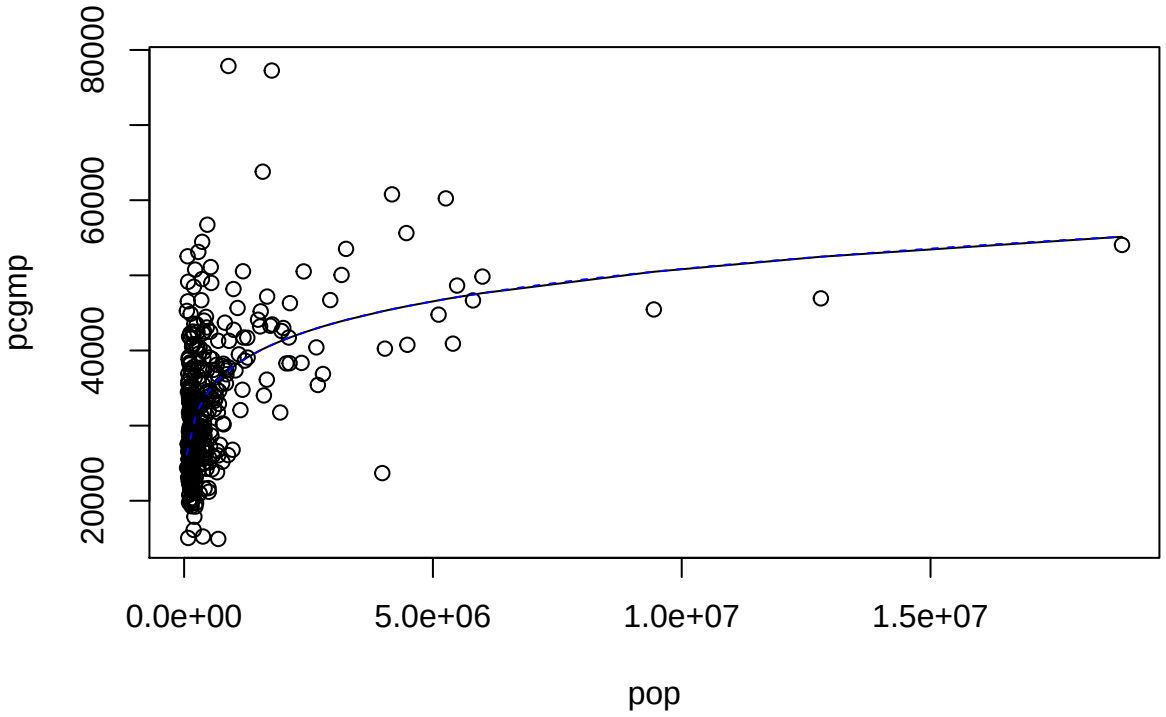
fit2: Fitting the Same Model with nls()

```
fit2 <- nls(pcgmp ~ y0 * pop^a, data = gmp, start = list(y0 = 5000, a = 0.1))
summary(fit2)
```

```
##
## Formula: pcgmp ~ y0 * pop^a
##
## Parameters:
##    Estimate Std. Error t value Pr(>|t|)
## y0 6.494e+03  8.565e+02   7.582 2.87e-13 ***
## a  1.277e-01  1.012e-02  12.612  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7886 on 364 degrees of freedom
##
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 1.789e-07
```

fit2: Fitting the Same Model with nls()

```
plot(pcgmp ~ pop, data = gmp)
pop.order <- order(gmp$pop)
lines(gmp$pop[pop.order], fitted(fit2)[pop.order])
curve(fit1$par[1] * x^fit1$par[2], add = TRUE, lty = "dashed", col = "blue")
```



Example: Multinomial

Roll dice n times; $n_1, \ldots n_6$ count the outcomes

Likelihood and log-likelihood:

$$L(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6) = \frac{n!}{n_1!n_2!n_3!n_4!n_5!n_6!} \prod_{i=1}^{6} \theta_i^{n_i} \ell(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6) = \log \frac{n!}{n_1!n_2!n_3!n_4!n_5!n_6!} + \sum_{i=1}^{6} n_i \log \theta_i$$

Optimize by taking the derivative and setting to zero:

$$\frac{\partial \ell}{\partial \theta_1} = \frac{n_1}{\theta_1} = 0 \therefore \ \theta_1 = \infty$$

We forgot that $\sum_{i=1}^{6} \theta_i = 1$

We could use the constraint to eliminate one of the variables

$$\theta_6 = 1 - \sum_{i=1}^{5} \theta_i$$

Then solve the equations

$$\frac{\partial \ell}{\partial \theta_i} = \frac{n_1}{\theta_i} - \frac{n_6}{1 - \sum_{j=1}^{5} \theta_j} = 0$$

BUT eliminating a variable with the constraint is usually messy

Lagrange multipliers

$$g(\theta) = c \;\Leftrightarrow\; g(\theta) - c = 0$$

Lagrangian:

$$\mathcal{L}(\theta, \lambda) = f(\theta) - \lambda(g(\theta) - c)$$

$= f$ when the constraint is satisfied

Now do *unconstrained* minimization over θ and λ:

$$\nabla_\theta \mathcal{L}|_{\theta^*,\lambda^*} = \nabla f(\theta^*) - \lambda^* \nabla g(\theta^*) = 0 \, \frac{\partial \mathcal{L}}{\partial \lambda}\bigg|_{\theta^*,\lambda^*} = g(\theta^*) - c = 0$$

optimizing **Lagrange multiplier** λ enforces constraint

More constraints, more multipliers

Try the dice again:

$$\mathcal{L} = \log \frac{n!}{\prod_i n_i!} + \sum_{i=1}^{6} n_i \log(\theta_i) - \lambda \left(\sum_{i=1}^{6} \theta_i - 1 \right) \; \frac{\partial \mathcal{L}}{\partial \theta_i}\bigg|_{\theta_i = \theta_i^*} = \frac{n_i}{\theta_i^*} - \lambda^* = 0 \frac{n_i}{\lambda^*} = \theta_i^* \sum_{i=1}^{6} \frac{n_i}{\lambda^*} = \sum_{i=1}^{6} \theta_i^* = 1 \lambda^* = \sum_{i=1}^{6} n_i \;\Rightarrow \theta_i^* = \frac{n_i}{\sum_{i=}^{6}}$$

Constrained minimum value is generally higher than the unconstrained

Changing the constraint level c changes θ^*, $f(\theta^*)$

$$\frac{\partial f(\theta^*)}{\partial c} = \frac{\partial \mathcal{L}(\theta^*, \lambda^*)}{\partial c} = \left[\nabla f(\theta^*) - \lambda^* \nabla g(\theta^*)\right] \frac{\partial \theta^*}{\partial c} - \left[g(\theta^*) - c\right] \frac{\partial \lambda^*}{\partial c} + \lambda^* = \lambda^*$$

λ^* = Rate of change in optimal value as the constraint is relaxed

λ^* = "Shadow price": How much would you pay for minute change in the level of the constraint

Inequality Constraints

What about an *inequality* constraint?

$$h(\theta) \leq d \;\Leftrightarrow\; h(\theta) - d \leq 0$$

The region where the constraint is satisfied is the **feasible set**

Roughly two cases:

1. Unconstrained optimum is inside the feasible set ⇒ constraint is **inactive**
2. Optimum is outside feasible set; constraint **is active**, **binds** or **bites**; *constrained* optimum is usually on the boundary

Add a Lagrange multiplier; $\lambda \neq 0 \Leftrightarrow$ constraint binds

Mathematical Programming

Older than computer programming...

Optimize $f(\theta)$ subject to $g(\theta) = c$ and $h(\theta) \leq d$

"Give us the best deal on f, keeping in mind that we've only got d to spend, and the books have to balance"

Linear programming

1. f, h both linear in θ
2. θ^* always at a corner of the feasible set

Example: Factory

Revenue: 13k per car, 27k per truck

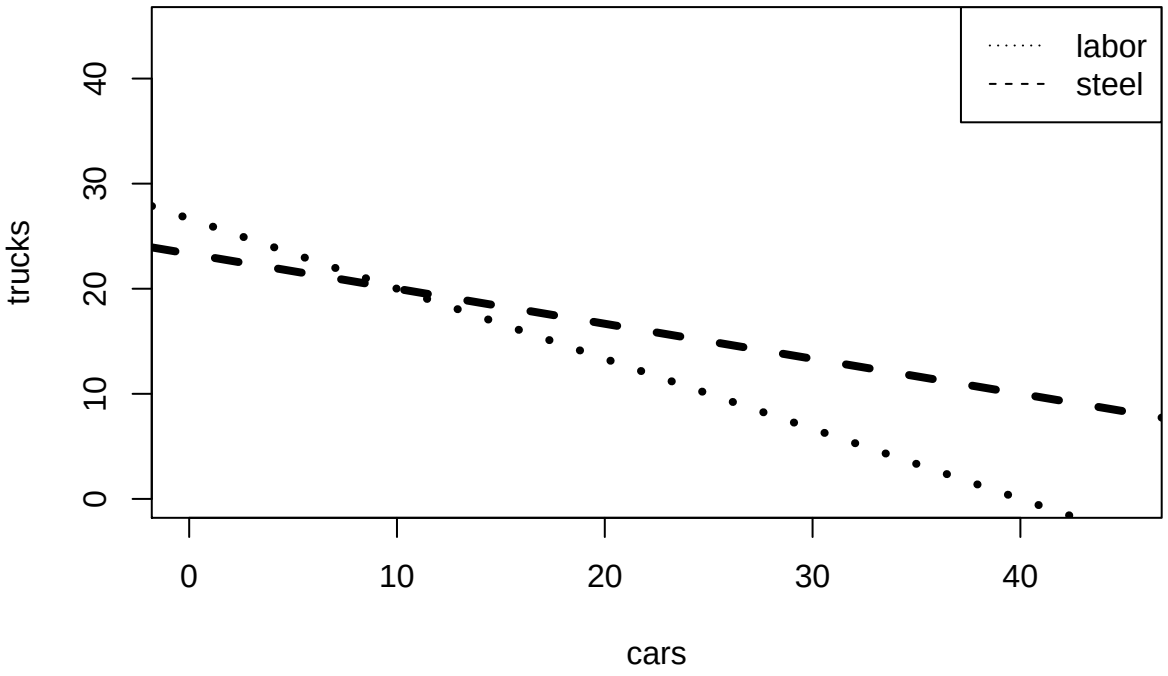
Constraints:

$$40 * \text{cars} + 60 * \text{trucks} < 1600 \text{hours} 1 * \text{cars} + 3 * \text{trucks} < 70 \text{tons}$$

Find the revenue-maximizing number of cars and trucks to produce

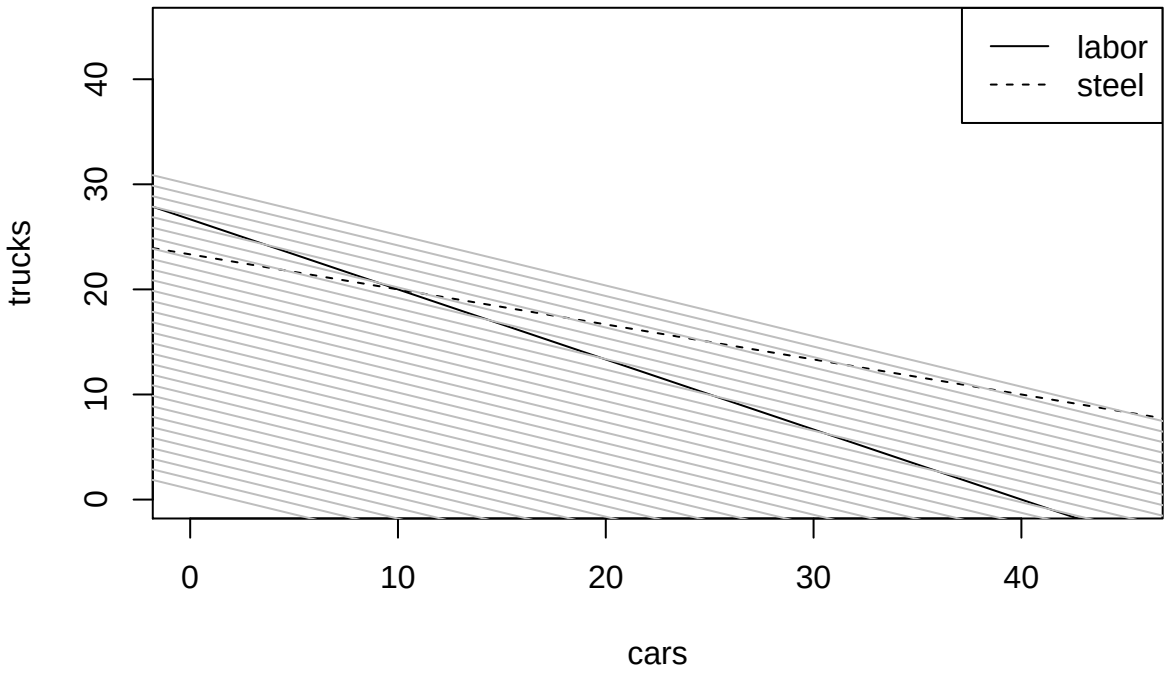
Example: Factory

The feasible region:



Example: Factory

The feasible region, plus lines of equal profit



More Complex Financial Problem

Given: expected returns $r_1, \ldots r_p$ among p financial assets, their $p \times p$ matrix of variances and covariances Σ

Find: the portfolio shares $\theta_1, \ldots \theta_n$ which maximizes expected returns

Such that: total variance is below some limit, covariances with specific other stocks or portfolios are below some limit
e.g., pension fund should not be too correlated with parent company

Expected returns $f(\theta) = r \cdot \theta$

Constraints: $\sum_{i=1}^{p} \theta_i = 1$, $\theta_i \geq 0$ (unless you can short)
Covariance constraints are linear in θ
Variance constraint is quadratic, over-all variance is $\theta^T \Sigma \theta$

Barrier Methods

(a.k.a. "interior point", "central path", etc.)

Having constraints switch on and off abruptly is annoying, especially with gradient methods

Fix $\mu > 0$ and try minimizing

$$f(\theta) - \mu \log (d - h(\theta))$$

"pushes away" from the barrier — more and more weakly as $\mu \to 0$

1. Initial θ in feasible set, initial μ
2. While ((not too tired) and (making adequate progress))
    a. Minimize $f(\theta) - \mu \log (d - h(\theta))$
    b. Reduce μ
3. Return final θ

R implementation

`constrOptim` implements the barrier method

Try this:

```
factory <- matrix(c(40, 1, 60, 3), nrow = 2, dimnames = list(c("labor", "steel"),
    c("car", "truck")))
available <- c(1600, 70)
names(available) <- rownames(factory)
prices <- c(car = 13, truck = 27)
revenue <- function(output) {
    return(-output %*% prices)
}
plan <- constrOptim(theta = c(5, 5), f = revenue, grad = NULL, ui = -factory,
    ci = -available, method = "Nelder-Mead")
plan$par
```

```
## [1]  9.999896 20.000035
```

`constrOptim` only works with constraints like $\mathbf{u}\theta \geq c$, so minus signs

Constraints vs. Penalties

$$\underset{\theta: h(\theta) \leq d}{\operatorname{argmin}} f(\theta) \Leftrightarrow \underset{\theta, \lambda}{\operatorname{argmin}} f(\theta) - \lambda(h(\theta) - d)$$

d doesn't matter for doing the second minimization over θ

We could just as well minimize

$$f(\theta) - \lambda h(\theta)$$

Constrained optimization	Penalized optimization
Constraint level d	Penalty factor λ

Statistical applications of penalization

Mostly you've seen unpenalized estimates (least squares, maximum likelihood)

Lots of modern advanced methods rely on penalties

- For when the direct estimate is too unstable
- For handling high-dimensional cases
- For handling non-parametrics

Ordinary least squares

No penalization; minimize MSE of linear function $\beta \cdot x$:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta \cdot x_i)^2 = \underset{\beta}{\operatorname{argmin}} MSE(\beta)$$

Closed-form solution if we can invert matrices:

$$\hat{\beta} = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y}$$

where $\mathbf{x}$ is the $n \times p$ matrix of x vectors, and $\mathbf{y}$ is the $n \times 1$ matrix of y values.

Ridge regression

Now put a penalty on the *magnitude* of the coefficient vector:

$$\tilde{\beta} = \underset{\beta}{\operatorname{argmin}}\, MSE(\beta) + \mu \sum_{j=1}^{p} \beta_j^2 = \underset{\beta}{\operatorname{argmin}}\, MSE(\beta) + \mu\|\beta\|_2^2$$

Penalizing β this way makes the estimate more *stable*; especially useful for

- Lots of noise
- Collinear data ($\mathbf{x}$ not of "full rank")
- High-dimensional, $p > n$ data (which implies collinearity)

This is called **ridge regression**, or **Tikhonov regularization**

Closed form:

$$\tilde{\beta} = (\mathbf{x}^T\mathbf{x} + \mu I)^{-1}\mathbf{x}^T\mathbf{y}$$

The Lasso

Put a penalty on the sum of coefficient's absolute values:

$$\beta^\dagger = \underset{\beta}{\operatorname{argmin}}\, MSE(\beta) + \lambda \sum_{j=1}^{p} |\beta_j| = \underset{\beta}{\operatorname{argmin}}\, MSE(\beta) + \lambda\|\beta\|_1$$

This is called **the lasso**

- Also stabilizes (like ridge)
- Also handles high-dimensional data (like ridge)
- Enforces **sparsity**: it likes to drive small coefficients exactly to 0

No closed form, but very efficient interior-point algorithms (e.g., `lars` package)

Spline smoothing

"Spline smoothing": minimize MSE of a smooth, nonlinear function, plus a penalty on curvature:

$$\hat{f} = \underset{f}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 + \int (f''(x))^2 dx$$

This fits smooth regressions without assuming any specific functional form

- Lets you check linear models
- Makes you wonder why you bother with linear models

Many different R implementations, starting with `smooth.spline`

How Big a Penalty?

Rarely know the constraint level or the penalty factor λ from on high

Lots of ways of picking, but often **cross-validation** works well

- Divide the data into parts
- For each value of λ, estimate the model on one part of the data
- See how well the models fit the other part of the data
- Use the λ which extrapolates best on average

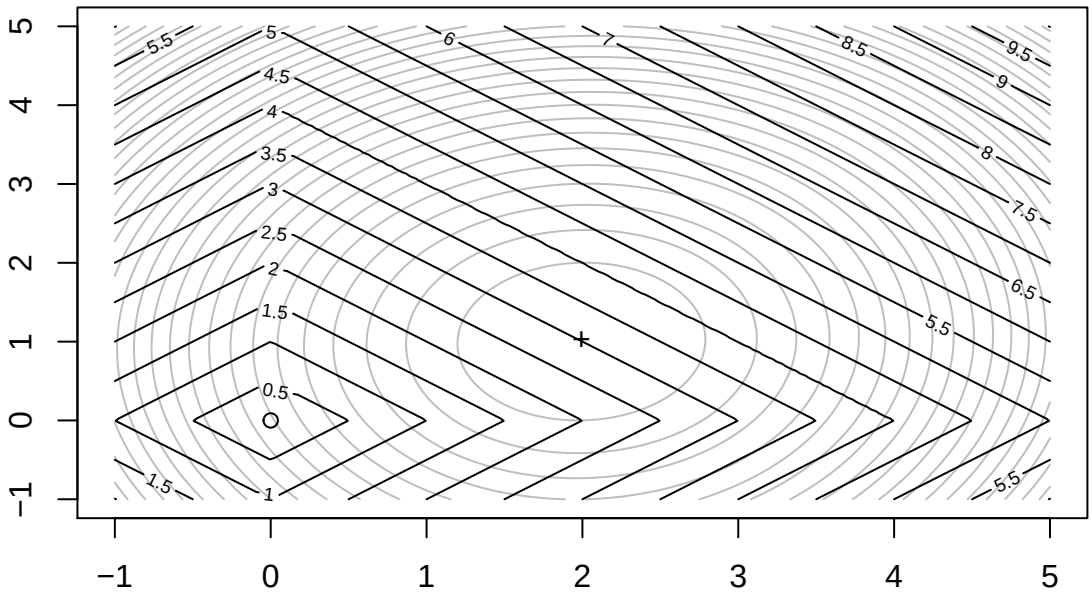
Summary

- Start with pre-built code like `optim` or `nls`, implement your own as needed
- Use Lagrange multipliers to turn constrained optimization problems into unconstrained but penalized ones
  - Optimal multiplier values are the prices we'd pay to weaken the constraints
- The nature of the penalty term reflects the sort of constraint we put on the problem
  - Shrinkage
  - Sparsity
  - Smoothness

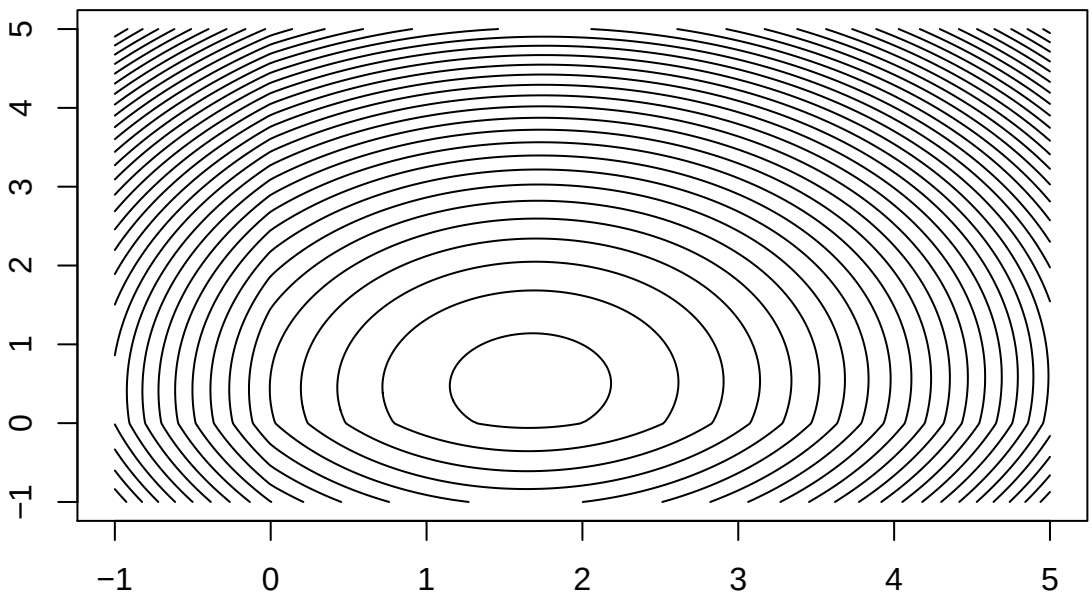
Example: Lasso

```
x <- matrix(rnorm(200), nrow = 100)
y <- (x %*% c(2, 1)) + rnorm(100, sd = 0.05)
mse <- function(b1, b2) {
    mean((y - x %*% c(b1, b2))^2)
}
coef.seq <- seq(from = -1, to = 5, length.out = 200)
m <- outer(coef.seq, coef.seq, Vectorize(mse))
l1 <- function(b1, b2) {
    abs(b1) + abs(b2)
}
l1.levels <- outer(coef.seq, coef.seq, l1)
ols.coefs <- coefficients(lm(y ~ 0 + x))
```

```
contour(x = coef.seq, y = coef.seq, z = m, drawlabels = FALSE, nlevels = 30,
    col = "grey", main = "Contours of MSE vs. Contours of L1")
contour(x = coef.seq, y = coef.seq, z = l1.levels, nlevels = 20, add = TRUE)
points(x = ols.coefs[1], y = ols.coefs[2], pch = "+")
points(0, 0)
```




```
contour(x = coef.seq, y = coef.seq, z = m + l1.levels, drawlabels = FALSE, nlevels = 30,
    main = "Contours of MSE+L1")
```



Bonus: Writing Our Own gradient()

- Suppose we didn't know about the `numDeriv` package..
- Use the simplest possible method: change `x` by some amount, find the difference in `f`, take the slope `method="simple"` option in `numDeriv::grad`
- Start with pseudo-code

```
gradient <- function(f,x,deriv.steps) {
  # not real code
  evaluate the function at x and at x+deriv.steps
  take slopes to get partial derivatives
  return the vector of partial derivatives
}
```

A naive implementation would use a `for` loop

```
gradient <- function(f, x, deriv.steps, ...) {
    p <- length(x)
    stopifnot(length(deriv.steps) == p)
    f.old <- f(x, ...)
    gradient <- vector(length = p)
    for (coordinate in 1:p) {
        x.new <- x
        x.new[coordinate] <- x.new[coordinate] + deriv.steps[coordinate]
        f.new <- f(x.new, ...)
        gradient[coordinate] <- (f.new - f.old)/deriv.steps[coordinate]
    }
    return(gradient)
}
```

Works, but it's so repetitive!

Better: use matrix manipulation and `apply`

```
gradient <- function(f, x, deriv.steps, ...) {
    p <- length(x)
    stopifnot(length(deriv.steps) == p)
    x.new <- matrix(rep(x, times = p), nrow = p) + diag(deriv.steps, nrow = p)
    f.new <- apply(x.new, 2, f, ...)
    gradient <- (f.new - f(x, ...))/deriv.steps
    return(gradient)
}
```

(clearer, and half as long)

- Presumes that `f` takes a vector and returns a single number
- Any extra arguments to `gradient` will get passed to `f`
- Check: Does this work when `f` is a function of a single number?
- Acts badly if `f` is only defined on a limited domain and we ask for the gradient somewhere near a boundary
- Forces the user to choose `deriv.steps`
- Uses the same `deriv.steps` everywhere, imagine $f(x) = x^2 \sin x$

. . . and so on through much of a first course in numerical analysis

Lecture 10: Simulations

Agenda

- Simulating from distributions
- Quantile transform method
- Rejection sampling

Simulation

Why simulate?

- We want to see what a probability model actually does
- We want to understand how our procedure works on a test case
- We want to use a partly-random procedure

All of these require drawing random variables from distributions

We have seen R has built in distributions: `beta`, `binom`, `cauchy`, `chisq`, `exp`, `f`, `gamma`, `geom`, `hyper`, `logis`, `lnorm`, `nbinom`, `norm`, `pois`, `t`, `tukey`, `unif`, `weibull`, `wilcox`, `signrank`

Every distribution that R handles has four functions.

- `p` for "probability", the cumulative distribution function (c. d. f.)
- `q` for "quantile", the inverse c. d. f.
- `d` for "density", the density function (p. f. or p. d. f.)
- `r` for "random", a random variable having the specified distribution
- Usually, R gets Uniform$(0, 1)$ random variates via a pseudorandom generator, e.g. the linear congruential generator
- Uses a sequence of Uniform$(0, 1)$ random variates to generate other distributions
- How?

Example: Binomial

- Suppose we want to generate a Binomial$(1, 1/3)$ using a $U \sim$ Uniform$(0, 1)$
- Consider the function $X^* = I(0 < u < 1/3)$, then

$$P(X^* = 1) = P(I(0 < u < 1/3 = 1) = P(u \in (0, 1/3)) = 1/3$$

  and $P(X^* = 1) = 2/3$
- Hence, $X^* \sim$ Binomial$(1, 1/3)$
- Two ways to extend this to Binomial$(n, 1/3)$

```
my.binom.1 <- function(n = 1, p = 1/3) {
    u <- runif(n)
    binom <- sum(u < p)
    return(binom)
}

my.binom.1(1000)
```

```
## [1] 337
```

```r
my.binom.1(1000, 0.5)
```

```
## [1] 502
```

```r
my.binom.2 <- function(n = 1, p = 1/3) {
    u <- runif(1)
    binom <- qbinom(u, size = n, prob = p)
    return(binom)
}

my.binom.2(1000)
```

```
## [1] 342
```

```r
my.binom.2(1000, 0.5)
```

```
## [1] 472
```

Quantile transform method

Given $U \sim \text{Uniform}(0, 1)$ and CDF F from a continuous distribution. Then $X = F^{-1}(U)$ is a random variable with CDF F.

Proof:

$$P(X \leq a) = P(F^{-1}(U) \leq a) = P(U \leq F(a)) = F(a)$$

- F^{-1} is the quantile function
- If we can generate uniforms and calculate quantiles, we can generate non-uniforms
- Also known as the Probability Integral Transform Method

Example: Exponential

Suppose $X \sim \text{Exp}(\beta)$. Then we have density

$$f(x) = \beta^{-1} e^{-x/\beta} I(0 < x < \infty)$$

and CDF

$$F(x) = 1 - e^{-x/\beta}$$

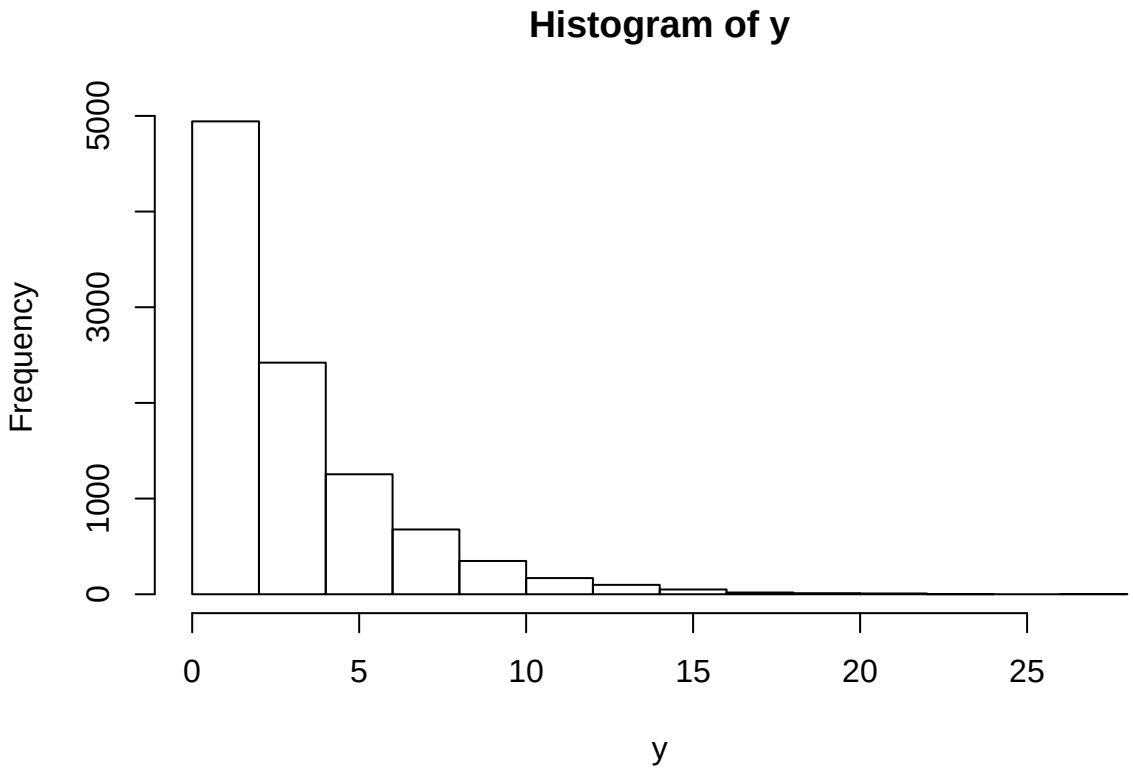
Also

$$y = 1 - e^{-x/\beta} \text{ iff } -x/\beta = \log(1 - y) \text{ iff } x = -\beta \log(1 - y).$$

Thus, $F^{-1}(y) = -\beta \log(1 - y)$.

So if $U \sim \text{Uniform}(0, 1)$, then $F^{-1}(u) = -\beta \log(1 - u) \sim \text{Exp}(\beta)$.

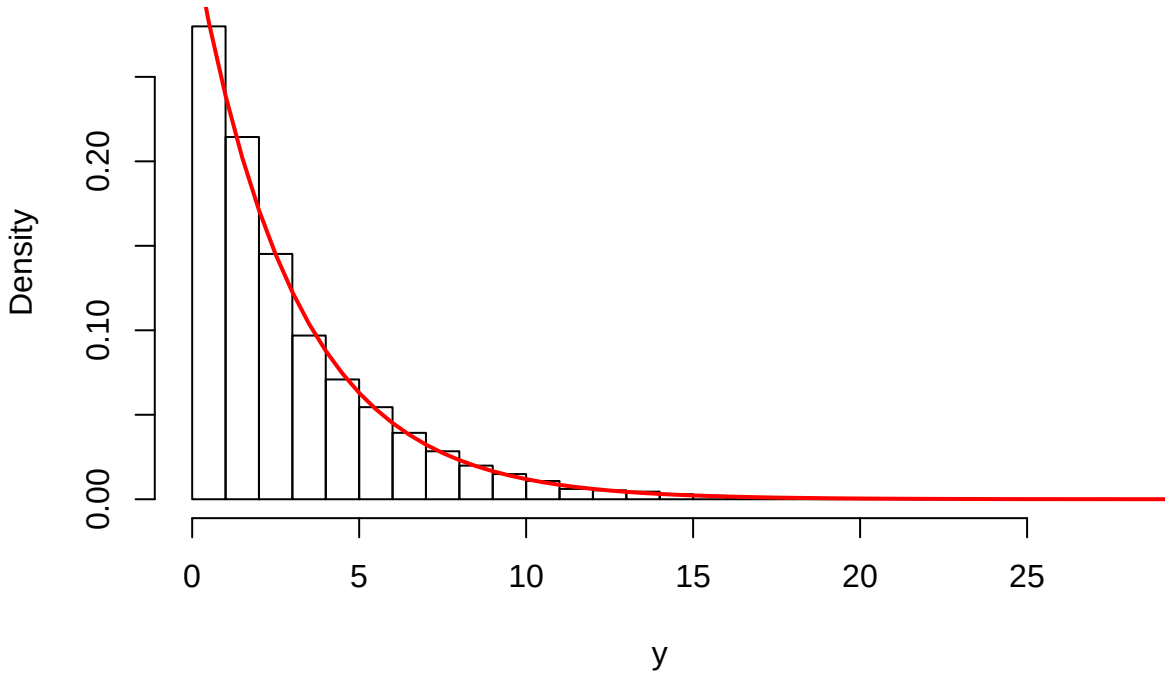
```r
x <- runif(10000)
y <- -3 * log(1 - x)
hist(y)
```




```
mean(y)
```

```
## [1] 2.993253
```

```
true.x <- seq(0, 30, 0.5)
true.y <- dexp(true.x, 1/3)
hist(y, freq = F, breaks = 30)
points(true.x, true.y, type = "l", col = "red", lw = 2)
```



Example: Gamma

- Remember that if $X_1, \ldots, X_n$ are IID Exp(β), then $\sum_{i=1}^{n} X_i \sim \Gamma(n, \beta)$
- Hence if we need a $\Gamma(\alpha, \beta)$ random variate and $\alpha \in \{1, 2, \ldots\}$, then take $U_1, \ldots, U_\alpha$ IID Uniform$(0, 1)$ and set

$$\sum_{i=1}^{\alpha} -\beta \log(1 - u_i) \sim \Gamma(\alpha, \beta)$$

- What if α is not an integer?

Quantile transform method

- Quantile functions often don't have closed form solutions or even nice numerical solutions
- But we know the probability density function â€" can we use that?

Rejection sampling

- The *accept-reject algorithm* is an indirect method of simulation
- Uses draws from a density $f_y(y)$ to get draws from $f_x(x)$
- Sampling from the wrong distribution and correcting it

Theorem: Let $X \sim f_x$ and $Y \sim f_y$ where the two densities have common support. Define

$$M = \sup_x \frac{f_x(x)}{f_y(x)}.$$

If $M < \infty$ then we can generate $X \sim f_x$ as follows, 1. Generate $Y \sim f_y$ and independently draw $U \sim \text{Uniform}(0,1)$ 2. If

$$u < \frac{f_x(y)}{M f_y(y)}$$

set $X = Y$; otherwise return to 1.

Exercise: Why is $M \geq 1$?

Proof:

$$\begin{aligned} P(X \leq x) &= P\left(Y \leq x \mid \text{STOP}\right) \\ &= P\left(Y \leq x \;\middle|\; u \leq \frac{f_x(y)}{M f_y(y)}\right) \\ &= \frac{P\left(Y \leq x, u \leq \frac{f_x(y)}{M f_y(y)}\right)}{P\left(u \leq \frac{f_x(y)}{M f_y(y)}\right)} \\ &= \frac{A}{B} \end{aligned}$$

Now, we have

$$\begin{aligned} A &= P\left(Y \leq x, u \leq \frac{f_x(y)}{M f_y(y)}\right) \\ &= E\left[P\left(Y \leq x, u \leq \frac{f_x(y)}{M f_y(y)}\right) \;\middle|\; y\right] \\ &= E\left[I(y \leq x)\frac{f_x(y)}{M f_y(y)}\right] \\ &= \int_{-\infty}^{\infty} I(y \leq x)\frac{f_x(y)}{M f_y(y)} f_y(y) dy \\ &= \frac{1}{M}\int_{-\infty}^{x} f_x(y) dy = \frac{F_x(x)}{M} \end{aligned}$$

Similarly, we have

$$\begin{aligned} B &= P\left(u \leq \frac{f_x(y)}{M f_y(y)}\right) \\ &= E\left[P\left(u \leq \frac{f_x(y)}{M f_y(y)}\right) \;\middle|\; y\right] \\ &= E\left[\frac{f_x(y)}{M f_y(y)}\right] \\ &= \int_{-\infty}^{\infty} \frac{f_x(y)}{M f_y(y)} f_y(y) dy \\ &= \frac{1}{M}\int_{-\infty}^{\infty} f_x(y) dy = \frac{1}{M} \end{aligned}$$

Hence,

$$\begin{aligned} P(X \leq x) &= \frac{A}{B} \\ &= \frac{\frac{F_x(x)}{M}}{\frac{1}{M}} = F_x(x) \end{aligned}$$

And the proof is complete. That is, $X \sim f_x$.

- Notice,

$$P(\text{STOP}) = B = P\left(u \leq \frac{f_x(y)}{M f_y(y)}\right) = \frac{1}{M}$$

- Thus the number of iterations until the algorithm stops is Geometric$(1/M)$
- Hence, the expected number of iterations until acceptance is M.

Example: Gamma

- Suppose we want to simulate $X \sim \Gamma(3/2, 1)$ with density

$$f_x(x) = \frac{2}{\pi}\sqrt{x}e^{-x}I(0 < x < \infty).$$

- Can use the accept-reject algorithm with a $\Gamma(n, 1)$ and $n \in \{1, 2, \dots\}$ since we know how to simulate this
- Then we have

$$\begin{aligned} M &= \sup x > 0 \frac{f_x(x)}{f_y(x)} \\ &= \sup x > 0 \frac{\frac{2}{\pi}\sqrt{x}e^{-x}}{\frac{1}{(n-1)!}x^{n-1}e^{-x}} \\ &= c \sup x > 0 x^{-n+3/2} = \infty \end{aligned}$$

since

$$n < 3/2 \text{ implies } x^{-n+3/2} \to \infty \text{ as } x \to \infty$$

and

$$n > 3/2 \text{ implies } x^{-n+3/2} \to \infty \text{ as } x \to 0$$

- Hence, we need to be a little more creative with our proposal distribution
- We could consider a mixture distribution. That is, if $f_1(z)$ and $f_2(z)$ are both densities and $p \in [0, 1]$. Then

$$pf_1(z) + (1-p)f_2(z)$$

is also a density

- Consider a proposal that is a mixture of a $\Gamma(1, 1) = \text{Exp}(1)$ and a $\Gamma(2, 1)$, i.e.

$$f_y(y) = \left[pe^{-y} + (1-p)ye^{-y}\right] I(0 < y < \infty)$$

Now, we have

$$\begin{aligned} M &= \sup x > 0 \frac{f_x(x)}{f_y(x)} \\ &= \sup x > 0 \frac{\frac{2}{\pi}\sqrt{x}e^{-x}}{pe^{-x} + (1-p)xe^{-x}} \\ &= \frac{2}{\pi} \sup x > 0 \frac{\sqrt{x}}{p + (1-p)x} \\ &= \frac{2}{\pi} \frac{1}{2\sqrt{p(1-p)}} \end{aligned}$$

Exercise: Prove the last line, i.e. maximize $h(x) = \frac{\sqrt{x}}{p+(1-p)x}$ for $x > 0$ or $\log h(x)$.

- Note that M is minimized when $p = 1/2$ so that $M_{1/2} = 2/\sqrt{\pi} \approx 1.1283$.
- Then the accept-reject algorithm to simulate $X \sim \Gamma(3/2, 1)$ is as follows

1. Draw $Y \sim f_y$ with
$$f_y(y) = \left[pe^{-y} + (1-p)ye^{-y}\right] I(0 < y < \infty)$$
and and independently draw $U \sim \text{Uniform}(0, 1)$

2. If

$$u < \frac{2}{\sqrt{\pi}} \frac{f_x(y)}{f_y(y)} = \frac{2\sqrt{y}}{1+y}$$

set $X = Y$; otherwise return to 1

Simulating from mixtures

- Write $f(z) = pf_1(z) + (1-p)f_2(z)$ as the marginal of the joint given by

$$f(z|w) = f_1(z)I(w = 1) + f_2(z)I(w = 0)$$

where $W \sim \text{Binomial}(1, p)$
- Thus to simulate from $f(z)$

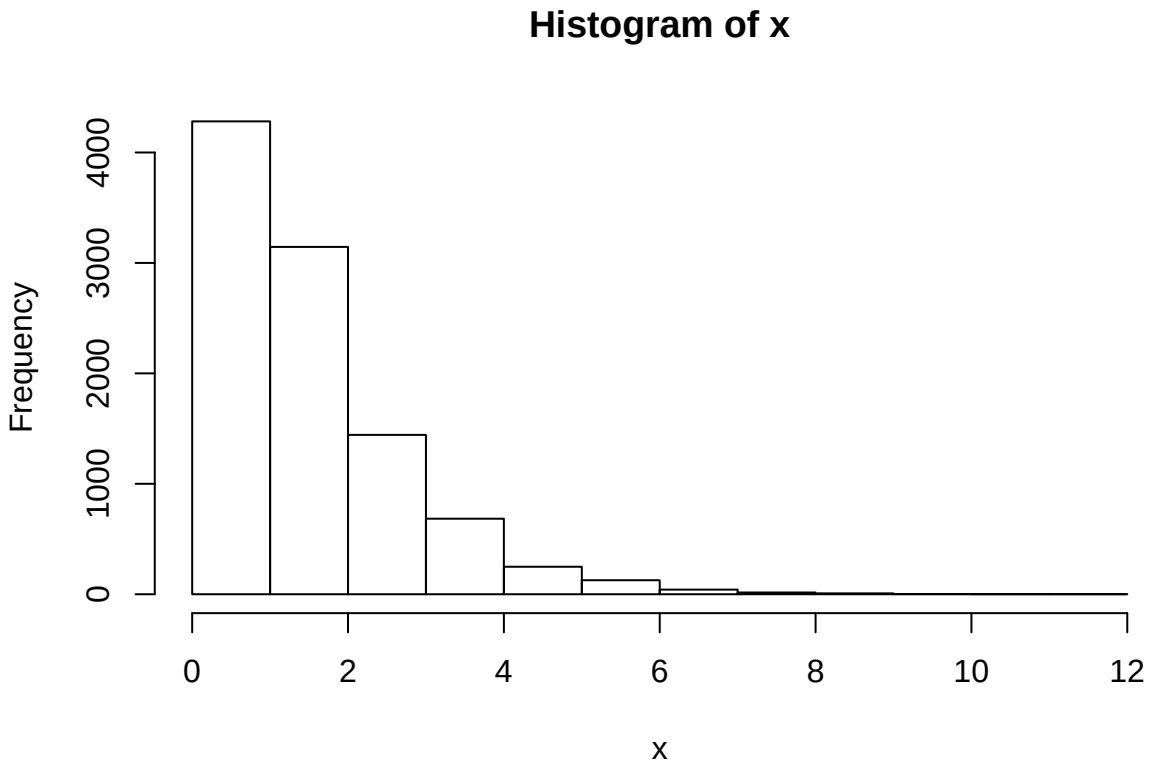
1. Draw $U \sim \text{Uniform}(0, 1)$
2. If $u < p$ take $Z \sim f_1(z)$; otherwise take $Z \sim f_2(z)$

- Exercise: Show $Z \sim f(z)$

Example: Gamma

```
ar.gamma <- function(n = 100) {
    x <- double(n)
    i <- 1
    while (i < (n + 1)) {
        u <- runif(1)
        if (u < 0.5) {
            y <- -1 * log(1 - runif(1))
        } else {
            y <- sum(-1 * log(1 - runif(2)))
        }
        u <- runif(1)
        temp <- 2 * sqrt(y)/(1 + y)
        if (u < temp) {
            x[i] <- y
            i <- i + 1
        }
    }
    return(x)
}
```

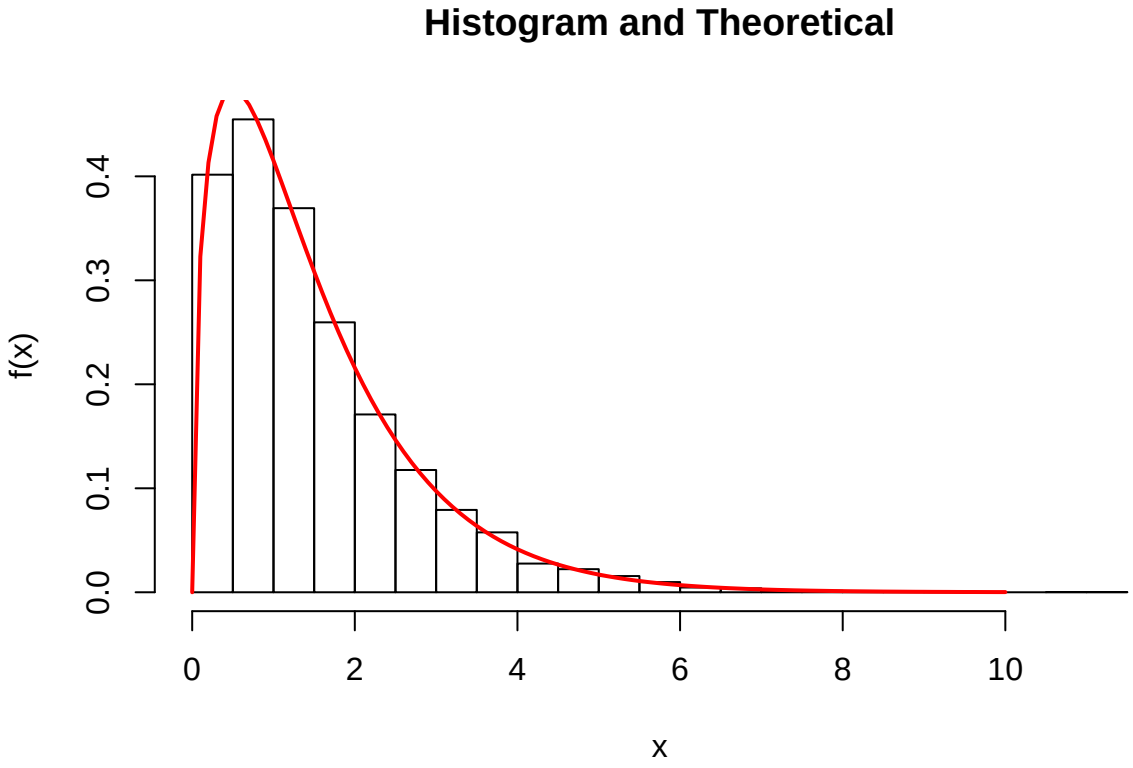
```
x <- ar.gamma(10000)
hist(x)
```




```
mean(x)
```

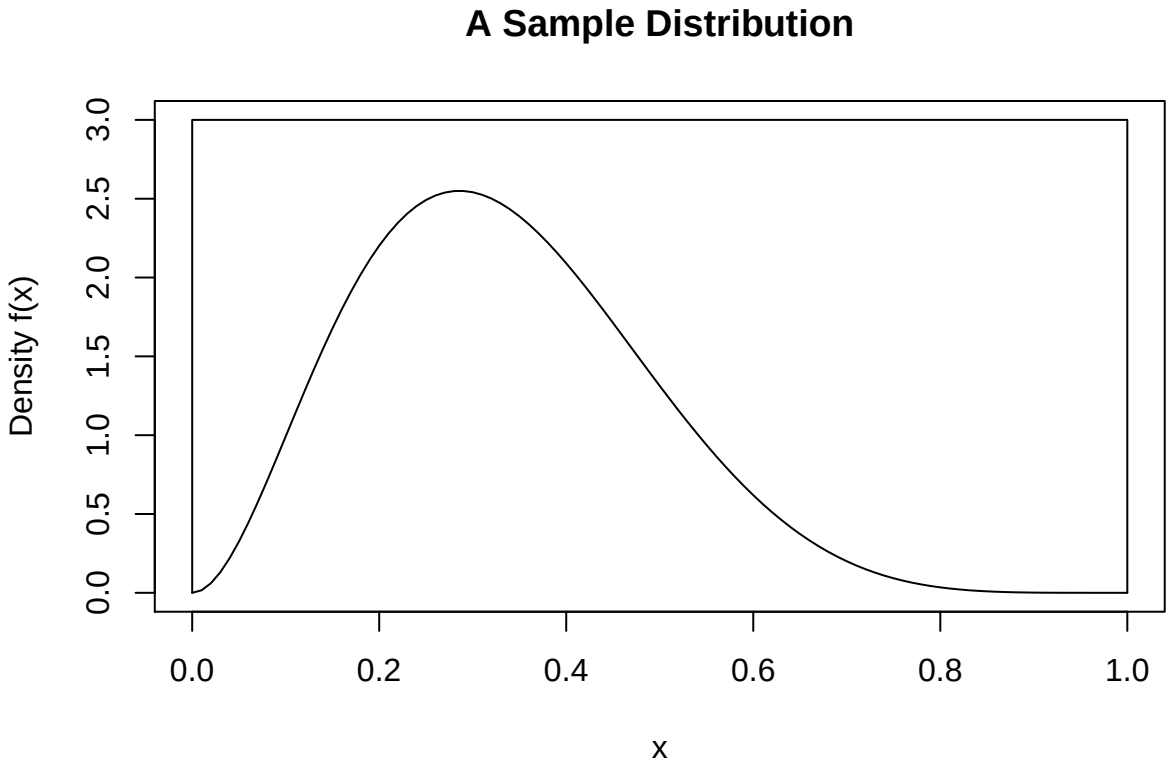
```
## [1] 1.496098
```

```
true.x <- seq(0, 10, 0.1)
true.y <- dgamma(true.x, 3/2, 1)
hist(x, freq = F, breaks = 30, xlab = "x", ylab = "f(x)", main = "Histogram and Theoretical")
points(true.x, true.y, type = "l", col = "red", lw = 2)
```



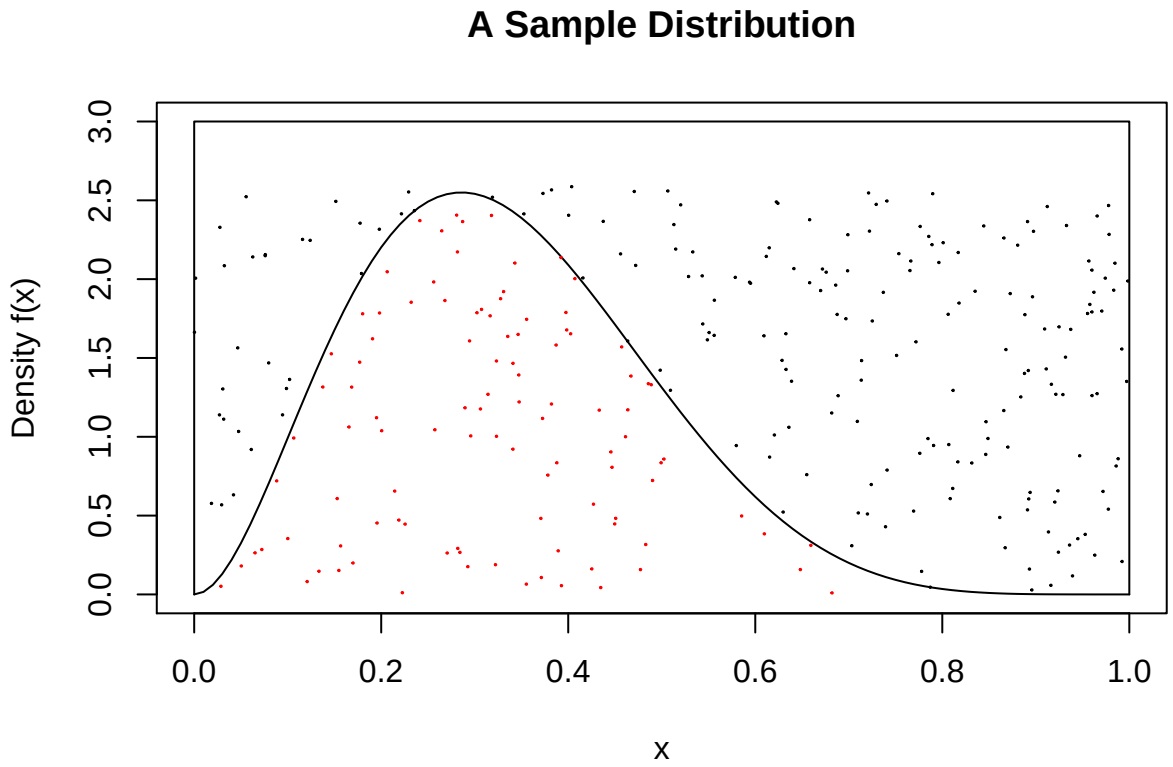
Example: Beta

Suppose the pdf f is zero outside an interval $[c, d]$, and $\leq M$ on the interval.



We know how to draw from uniform distributions in any dimension. Do it in two:

```
x1 <- runif(300, 0, 1)
y1 <- runif(300, 0, 2.6)
selected <- y1 < dbeta(x1, 3, 6)
```




```
mean(selected)
```

```
## [1] 0.3433333
```

```
accepted.points <- x1[selected]
mean(accepted.points < 0.5)
```

```
## [1] 0.9417476
```

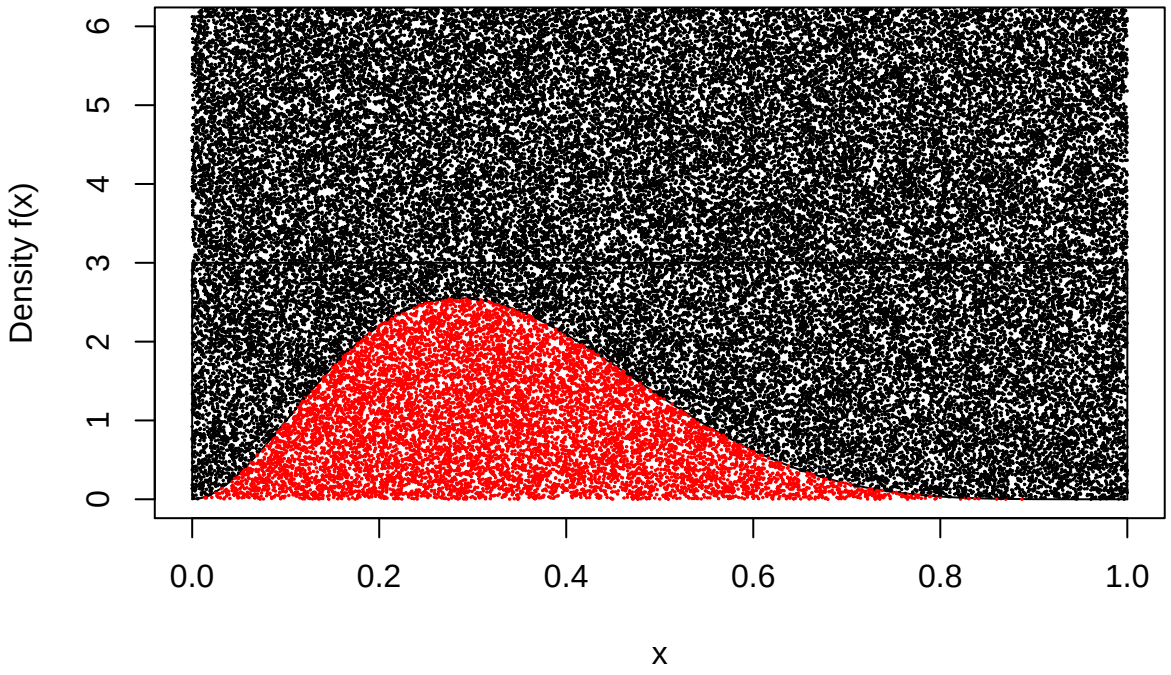
```
pbeta(0.5, 3, 6)
```

```
## [1] 0.8554688
```

For this to work efficiently, we have to cover the target distribution with one that sits close to it.

```
x2 <- runif(1e+05, 0, 1)
y2 <- runif(1e+05, 0, 10)
selected <- y2 < dbeta(x2, 3, 6)
mean(selected)
```

```
## [1] 0.10167
```



Alternatives

- Squeezed rejection sampling may help if evaluating f is expensive
- Adaptive rejection sampling may help generate an envelope
- ...

Box-Muller

- **Box-Muller transformation** transform generates pairs of independent, standard normally distributed random numbers, given a source of uniformly distributed random numbers
- Let $U \sim \text{Uniform}(0,1)$ and $V \sim \text{Uniform}(0,1)$ and set

$$R = \sqrt{-2 \log U} \qquad \text{and} \qquad \theta = 2\pi V$$

- Then the following transformation yields two independent normal random variates

$$X = R\cos(\theta) \qquad \text{and} \qquad Y = R\sin(\theta)$$

Summary

- Can transform uniform draws into other distributions when we can compute the distribution function
- Quantile method when we can invert the CDF
- The rejection method if all we have is the density
- Basic R commands encapsulate a lot of this for us
- Optimized algorithms based on distribution and parameter values

Lecture 11: Monte Carlo Simulations

Agenda

- Ordinary Monte Carlo
- Examples
- Monte Carlo integration
- Bootstrap
- Toy collector exercise

Ordinary Monte Carlo

The "Monte Carlo method" refers to the theory and practice of learning about probability distributions by simulation rather than calculus. In ordinary Monte Carlo (OMC) we use *IID* simulations from the distribution of interest. Suppose $X_1, X_2, \ldots$ are *IID* simulations from some distribution, and suppose we want to know an expectation

$$\theta = E\left[Y_1\right] = E\left[g(X_1)\right].$$

The law of large numbers (LLN) then says

$$\bar{y}_n = \frac{1}{n}\sum_{i=1}^{n} Y_i = \frac{1}{n}\sum_{i=1}^{n} g(X_i)$$

converges in probability to θ.

The central limit theorem (CLT) says

$$\frac{\sqrt{n}(\bar{y}_n - \theta)}{\sigma} \xrightarrow{d} \mathrm{N}(0,1).$$

That is, for sufficiently large n,

$$\bar{y}_n \sim \mathrm{N}(\theta, \sigma^2/n).$$

Further, we can estimate the standard error $\sigma/\sqrt{n}$ with $s_n/\sqrt{n}$ where s_n is the sample standard deviation.

We can also use the CLT form a confidence interval with

$$Pr(\bar{y}_n - 1.96 s_n/\sqrt{n} < \mathrm{E}Y_1 < \bar{y}_n + 1.96 s_n/\sqrt{n}) \approx 0.95.$$

Or we could simulate until a half-width (or width) of this confidence interval is sufficiently small, say less than $\epsilon > 0$. That is, simulate until

$$1.96 s_n/\sqrt{n} < \epsilon.$$

The theory of OMC is just the theory of frequentist statistical inference. The only differences are that

- the "data" $X_1, \ldots, X_n$ are computer simulations rather than measurements on objects in the real world
- the "sample size" n is the number of computer simulations rather than the size of some real world data
- the unknown parameter θ is in principle completely known, given by some integral, which we are unable to do.

Everything works just the same when the data $X_1, X_2, \ldots$, which are computer simulations are vectors. But the functions of interest $g(X_1), g(X_2), \ldots$ are scalars.

OMC works great, but it can be very difficult to simulate IID simulations of random variables or random vectors whose distribution is not brand name distributions

Approximating the Binomial

Suppose we flip a coin 10 times and we want to know the probability of getting more than 3 heads. This is trivial for the Binomial distribution, which we'll ignore.

```
runs <- 10000
one.trial <- function() {
    sum(sample(c(0, 1), 10, replace = T)) > 3
}
mc.binom <- sum(replicate(runs, one.trial()))/runs
mc.binom
```

```
## [1] 0.8346
```

```
pbinom(3, 10, 0.5, lower.tail = FALSE)
```

```
## [1] 0.828125
```

Exercise: Program this example and estimate the Monte Carlo standard error

Aproximating π

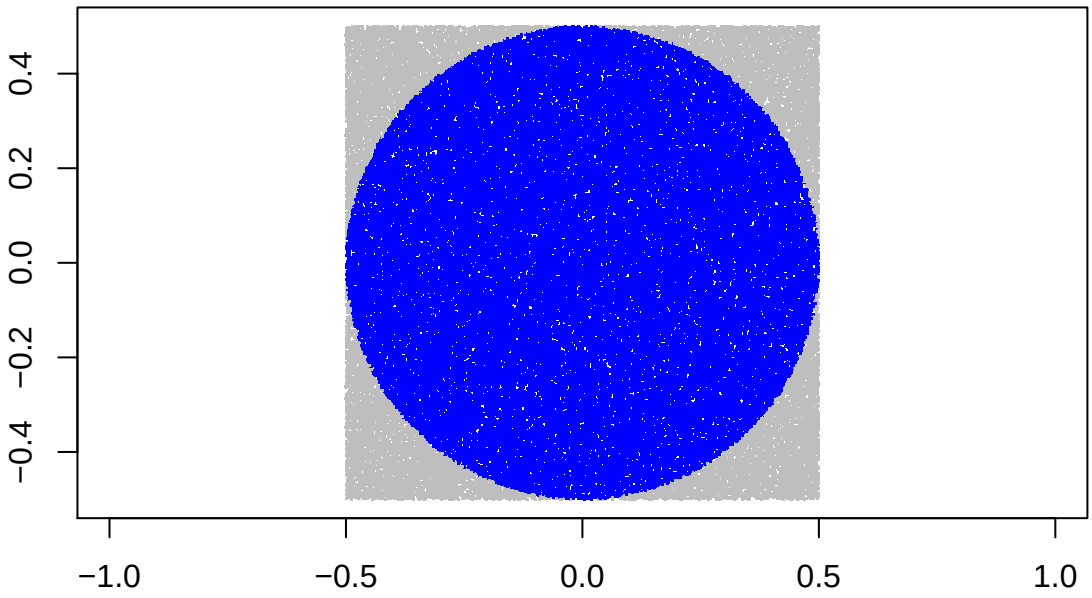
- Area of a circle is πr^2
- If we draw a square containing that circle its area will be $4r^2$
- So the ratio of the area of the circle to the area of the square is

$$\frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

- Given this fact, we can empirically determine the ratio of the area of the circle to the area of the square we can simply multiply this number by 4 and we'll get our approximation of π.
- How?
- Randomly sample x and y values on the unit square centered at 0
- If $x^2 + y^2 \leq .5^2$ then the point is in the circle
- The ratio of points in the circle multiplied by 4 is our estimate of π

```
runs <- 1e+05
xs <- runif(runs, min = -0.5, max = 0.5)
ys <- runif(runs, min = -0.5, max = 0.5)
in.circle <- xs^2 + ys^2 <= 0.5^2
mc.pi <- (sum(in.circle)/runs) * 4
```

```
plot(xs, ys, pch = ".", col = ifelse(in.circle, "blue", "grey"), xlab = "",
    ylab = "", asp = 1, main = paste("MC Approximation of Pi =", mc.pi))
```



Example: Integration

Let $X \sim \Gamma(3/2, 1)$, i.e.

$$f(x) = \frac{2}{\sqrt{\pi}} \sqrt{x} e^{-x} I(x > 0).$$

Suppose we want to find

$$\begin{aligned} \theta &= \mathrm{E}\left[\frac{1}{(X+1)\log(X+3)}\right] \\ &= \int_0^\infty \frac{1}{(x+1)\log(x+3)} \frac{2}{\sqrt{\pi}} \sqrt{x} e^{-x} dx. \end{aligned}$$

- The expectation (or integral) θ is intractable, we don't know how to compute it analytically
- Further, suppose we want to estimate this quantity such that a 95% CI length is less than 0.002

```
n <- 1000
x <- rgamma(n, 3/2, scale = 1)
mean(x)
```

```
## [1] 1.51416
```

```
y <- 1/((x + 1) * log(x + 3))
est <- mean(y)
est
```

```
## [1] 0.351348
```

```
mcse <- sd(y)/sqrt(length(y))
interval <- est + c(-1, 1) * 1.96 * mcse
interval
```

```
## [1] 0.3398184 0.3628777
```

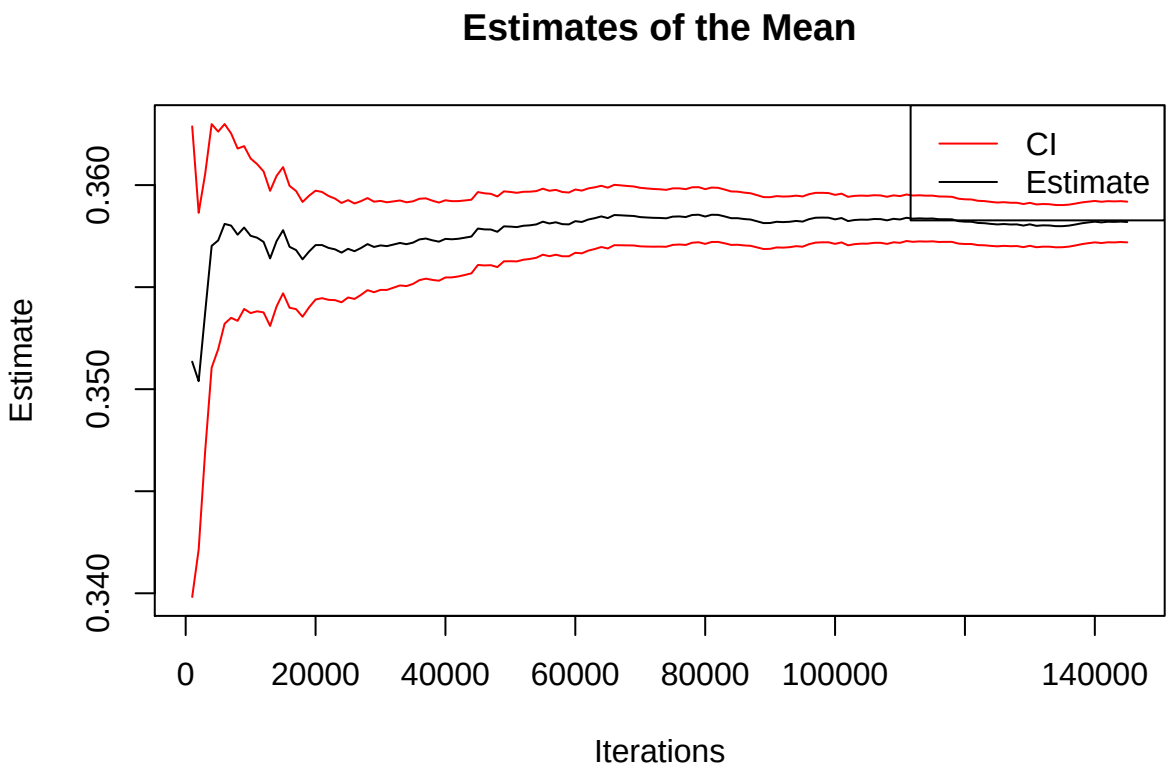
Example: Sequential stopping rule

```
eps <- 0.002
len <- diff(interval)
plotting.var <- c(est, interval)
while (len > eps) {
    new.x <- rgamma(n, 3/2, scale = 1)
    new.y <- 1/((new.x + 1) * log(new.x + 3))
    y <- cbind(y, new.y)
    est <- mean(y)
    mcse <- sd(y)/sqrt(length(y))
    interval <- est + c(-1, 1) * 1.96 * mcse
    len <- diff(interval)
    plotting.var <- rbind(plotting.var, c(est, interval))
}
list(interval, length(y))
```

```
## [[1]]
## [1] 0.3571937 0.3591882
##
## [[2]]
## [1] 145000
```

```
temp <- seq(1000, length(y), 1000)
```

```
plot(temp, plotting.var[, 1], type = "l", ylim = c(min(plotting.var), max(plotting.var)),
    main = "Estimates of the Mean", xlab = "Iterations", ylab = "Estimate")
points(temp, plotting.var[, 2], type = "l", col = "red")
points(temp, plotting.var[, 3], type = "l", col = "red")
legend("topright", legend = c("CI", "Estimate"), lty = c(1, 1), col = c(2, 1))
```



High-dimensional examples

- FiveThirtyEight's Election Forecast
- FiveThirtyEight's NBA Predictions
- Vanguard's Retirement Nest Egg Calculator
- Minitab's Monte Carlo Simulation Software for Manufacturing Engineers
- Fisher's Exact Test in R

Permutations with `sample()`

- `sample()` is powerful – it works on any object that has a defined `length()`.
- Permutations

```
sample(5)
```

```
## [1] 5 2 1 4 3
```

```
sample(1:6)
```

```
## [1] 1 2 5 6 3 4
```

```
replicate(3, sample(c("Curly", "Larry", "Moe", "Shemp")))
```

```
##      [,1]    [,2]    [,3]
## [1,] "Curly" "Shemp" "Moe"
## [2,] "Moe"   "Curly" "Larry"
## [3,] "Larry" "Larry" "Curly"
## [4,] "Shemp" "Moe"   "Shemp"
```

Resampling with `sample()`

Resampling from any existing distribution gives **bootstrap** estimators

```
bootstrap.resample <- function(object) sample(object, length(object), replace = TRUE)
replicate(5, bootstrap.resample(6:10))
```

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

Recall: the *jackknife* removed one point from the sample and recalculated the statistic of interest. Here we resample the same length with replacement.

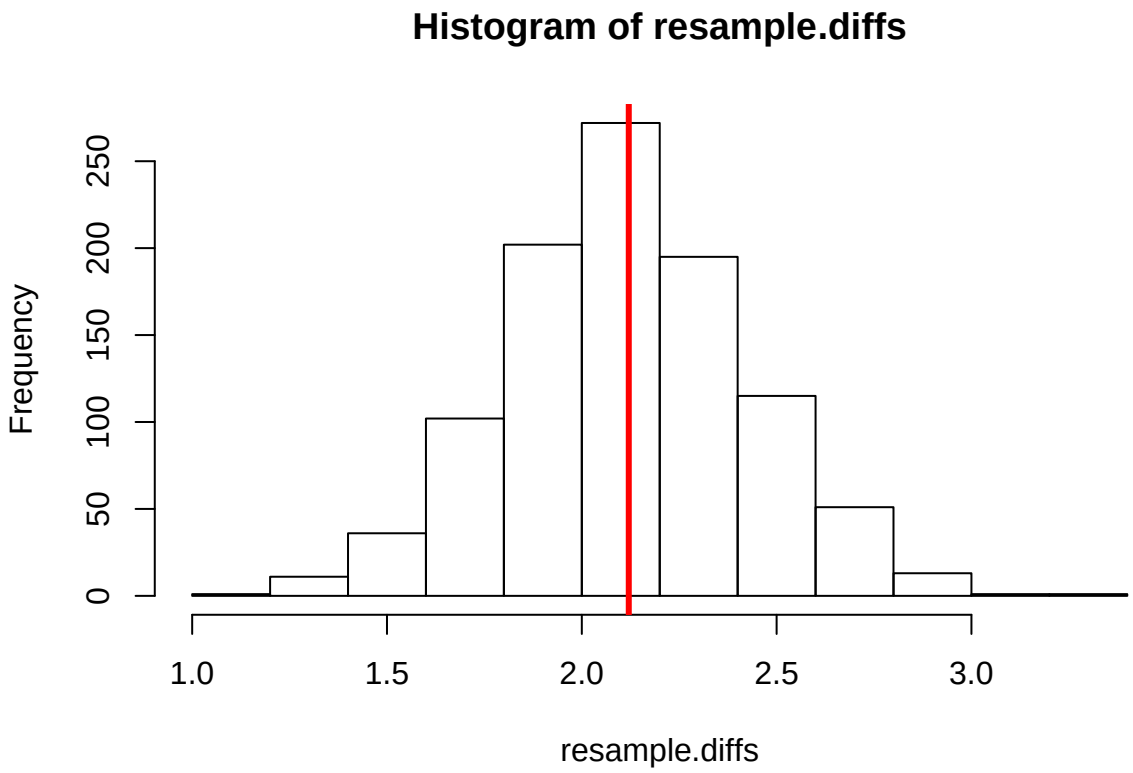
Bootstrap test

The 2-sample `t`-test checks for differences in means according to a known null distribution. Let's resample and generate the sampling distribution under the bootstrap assumption:

```
library(MASS)
diff.in.means <- function(df) {
    mean(df[df$Sex == "M", "Hwt"]) - mean(df[df$Sex == "F", "Hwt"])
}
```

```
resample.diffs <- replicate(1000, diff.in.means(cats[bootstrap.resample(1:nrow(cats)),
    ]))
```

```
hist(resample.diffs)
abline(v = diff.in.means(cats), col = 2, lwd = 3)
```



Summary

- Ordinary Monte Carlo
- Repeated random sampling to obtain numerical results
- Using randomness to solve problems
- Most useful when it is difficult or impossible to use other approaches
- Can you solve The Riddler?

Exercise: Toy Collector

Children (and some adults) are frequently enticed to buy breakfast cereal in an effort to collect all the action figures. Assume there are 15 action figures and each cereal box contains exactly one with each figure being equally likely.

- Find the expected number of boxes needed to collect all 15 action figures.
- Find the standard deviation of the number of boxes needed to collect all 15 action figures.
- Now suppose we no longer have equal probabilities, instead let

Figure	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Probability	.2	.1	.1	.1	.1	.1	.05	.05	.05	.05	.02	.02	.02	.02	.02

- Estimate the expected number of boxes needed to collect all 15 action figures.
- What is the uncertainty of your estimate?
- What is the probability you bought more than 300 boxes? 500 boxes? 800 boxes?

Lecture 12: Bootstrap

Agenda

- Toy collector solution
- Plug-In and the Bootstrap
- Nonparametric and Parametric Bootstraps
- Examples

Exercise: Toy Collector

Children (and some adults) are frequently enticed to buy breakfast cereal in an effort to collect all the action figures. Assume there are 15 action figures and each cereal box contains exactly one with each figure being equally likely.

- Find the expected number of boxes needed to collect all 15 action figures.
- Find the standard deviation of the number of boxes needed to collect all 15 action figures.
- Now suppose we no longer have equal probabilities, instead let

Figure	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Probability	.2	.1	.1	.1	.1	.1	.05	.05	.05	.05	.02	.02	.02	.02	.02

- Estimate the expected number of boxes needed to collect all 15 action figures.
- What is the uncertainty of your estimate?
- What is the probability you bought more than 300 boxes? 500 boxes? 800 boxes?
- Consider the probability of the "new toy" given we already have i toys
- Then

$$P(\text{New Toy}|i) = \frac{15-i}{15}$$

- Then since each box is independent, our waiting time until a "new toy" is a geometric random variable
- The mean is

$$\frac{15}{15} + \frac{15}{14} + \cdots + \frac{15}{1} \approx 49.77$$

- The variance is

$$\frac{15(1-15/15)}{15} + \frac{15(1-14/15)}{14} + \cdots + \frac{15(1-1/15)}{1} \approx 34.77$$

  with standard deviation 5.90

```
prob.table <- c(0.2, 0.1, 0.1, 0.1, 0.1, 0.1, 0.05, 0.05, 0.05, 0.05, 0.02,
    0.02, 0.02, 0.02, 0.02)
boxes <- seq(1, 15)
box.count <- function(prob = prob.table) {
    check <- double(length(prob))
    i <- 0
    while (sum(check) < length(prob)) {
        x <- sample(boxes, 1, prob = prob)
        check[x] <- 1
        i <- i + 1
    }
    return(i)
}
```

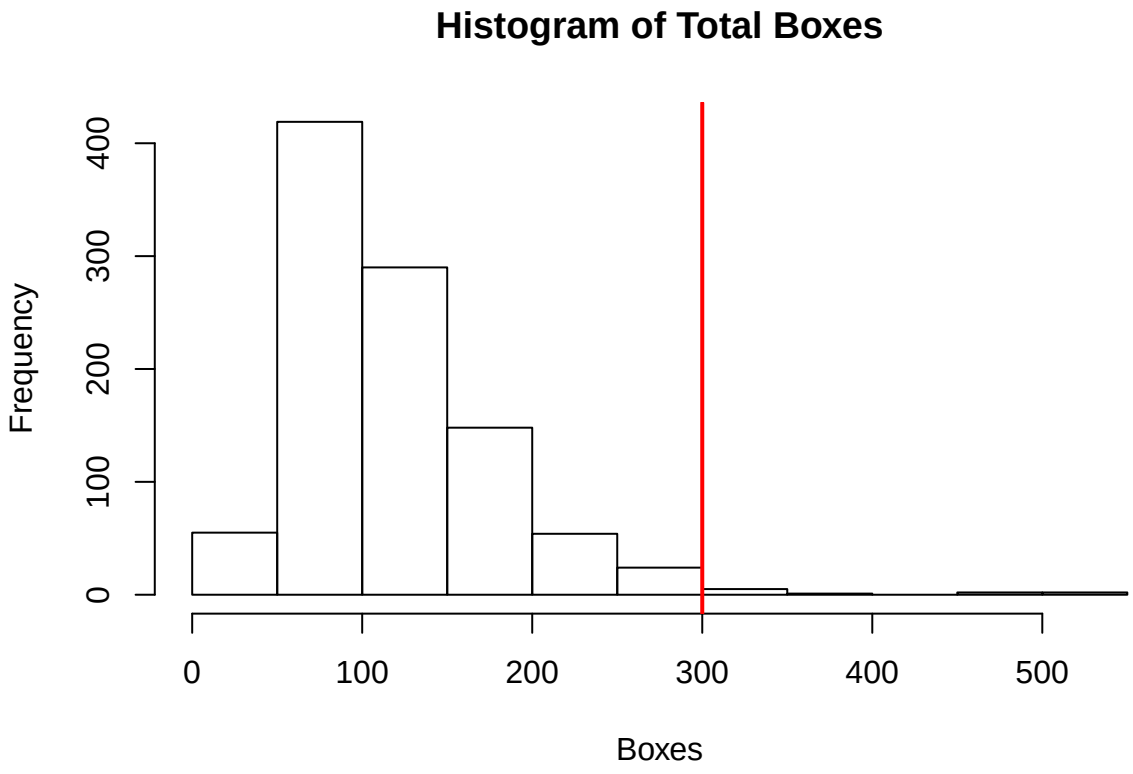
```
trials <- 1000
sim.boxes <- double(trials)
for (i in 1:trials) {
    sim.boxes[i] <- box.count()
}
est <- mean(sim.boxes)
mcse <- sd(sim.boxes)/sqrt(trials)
interval <- est + c(-1, 1) * 1.96 * mcse
est
```

```
## [1] 117.588
```

```
interval
```

```
## [1] 113.8997 121.2763
```

```
hist(sim.boxes, main = "Histogram of Total Boxes", xlab = "Boxes")
abline(v = 300, col = "red", lwd = 2)
```



Plug-In and the Bootstrap

- The worst mistake one can make in statistics is to confuse the sample and the population or to confuse estimators and parameters
- That is, $\hat{\theta}$ is not θ
- Plug-in principle seems to say the opposite
  - Sometimes it is okay to just plug in an estimate for an unknown parameter
  - In particular, okay to plug in a consistent estimator of the asymptotic variance of a parameter in forming asymptotic confidence intervals for that parameter
- So it is a terrible mistake to confuse a parameter of interest and an estimator for it, but it may not be a mistake to ignore the difference between a nuisance parameter and an estimator for it

- The "bootstrap" is a cute name for a vast generalization of the plug-in principle
- The name comes from the cliche "pull oneself up by one's bootstraps" which although it describes a literal impossibility actually means succeed by one's own efforts
- In statistics, the hint of impossibility is part of the flavor
- Seems problematic, but it (usually) works

Nonparametric Bootstrap

- The bootstrap comes in two flavors, parametric and nonparametric
- Theory of the nonparametric bootstrap is all above the level of this course, so we give a non-theoretical explanation
- The nonparametric bootstrap, considered non-theoretically, is just an analogy

World	Real	Bootstrap
true distribution	F	$\hat{F}_n$
data	$X_1, \ldots, X_n$ IID F	$X_1^*, \ldots, X_n^*$ IID $\hat{F}_n$
empirical distribution	$\hat{F}_n$	F_n^*
parameter	$\theta = t(F)$	$\hat{\theta}_n = t(\hat{F}_n)$
estimator	$\hat{\theta}_n = t(\hat{F}_n)$	$\theta_n^* = t(F_n^*)$
error	$\hat{\theta}_n - \theta$	$\theta_n^* - \hat{\theta}_n$
standardized error	$\frac{\hat{\theta}_n - \theta}{s(\hat{F}_n)}$	$\frac{\theta_n^* - \hat{\theta}_n}{s(F_n^*)}$

Notation $\theta = t(F)$ means θ is some function of the true unknown distribution

- The notation $X_1^*, \ldots, X_n^*$ IID $\hat{F}_n$ means $X_1^*, \ldots, X_n^*$ are independent and identically distributed from the empirical distribution of the real data
- Sampling from the empirical distribution is just like sampling from a finite population, where the "population" is the real data $X_1, \ldots, X_n$
  - To be IID sampling must be with replacement
  - $X_1^*, \ldots, X_n^*$ are a sample with replacement from $X_1, \ldots, X_n$
  - Called resampling
- We want to know the sampling distribution of $\hat{\theta}_n$ or $\hat{\theta}_n - \theta$ or $\frac{\hat{\theta}_n - \theta}{s(\hat{F}_n)}$
- This sampling distribution depends on the true unknown distribution F of the real data
- May be very difficult or impossible to calculate theoretically
- Even asymptotic approximation may be difficult, if the parameter $\theta = t(F)$ is a sufficiently complicated function of the true unknown F
- The statistical theory we have covered is quite amazing in what it does, but there is a lot it doesn't do
- In the "bootstrap world" everything is known, $\hat{F}_n$ plays the role of the true unknown distribution and $\hat{\theta}_n$ plays the role of the true unknown parameter value
- The sampling distribution of θ_n^* or $\theta_n^* - \hat{\theta}_n$ or $\frac{\theta_n^* - \hat{\theta}_n}{s(F_n^*)}$ may still be difficult to calculate theoretically, but it can always be "calculated" by simulation.
- Much folklore about the bootstrap is misleading
- The bootstrap is large sample, approximate, asymptotic
  - It is not an exact method

- The bootstrap analogy works when the empirical distribution $\hat{F}_n$ is close to the true unknown distribution F
  - Usually the case when the sample size n is large and not otherwise

Bootstrap Percentile Intervals

- Simplest method of making confidence intervals for the unknown parameter is to take $\alpha/2$ and $1 - \alpha/2$ quantiles of the bootstrap distribution of the estimator θ_n^* as endpoints of the 100(1 - α)% confidence interval
- Percentile method only makes sense when there is a symmetrizing transformation (some function of $\hat{\theta}$ has an approximately symmetric distribution with the center of symmetry being the true unknown parameter value θ)
- The symmetrizing transformation does not have to be known, but it does have to exist

Parametric Bootstrap

- The parametric bootstrap is just like the nonparametric bootstrap except for one difference in the analogy
- We use a parametric model $F_{\hat{\theta}_n}$ rather than the empirical distribution $\hat{F}_n$ as the analog of the true unknown distribution in the bootstrap world

World	Real	Bootstrap
parameter	θ	$\hat{\theta}_n$
true distribution	F_θ	$F_{\hat{\theta}_n}$
data	$X_1, \ldots, X_n$ IID F_θ	$X_1^*, \ldots, X_n^*$ IID $F_{\hat{\theta}_n}$
estimator	$\hat{\theta}_n = t(X_1, \ldots, X_n)$	$\theta_n^* = t(X_1^*, \ldots, X_n^*)$
error	$\hat{\theta}_n - \theta$	$\theta_n^* - \hat{\theta}_n$
standardized error	$\frac{\hat{\theta}_n - \theta}{s(X_1, \ldots, X_n)}$	$\frac{\theta_n^* - \hat{\theta}_n}{s(X_1^*, \ldots, X_n^*)}$

- Simulation from the parametric model $F_{\hat{\theta}_n}$ not analogous to finite population sampling and does not resample the data like the nonparametric bootstrap does
- Instead we simulate the parametric model
- May be easy (when R has a function to provide such random simulations) or difficult

Nonparametric versus Parametric

- The nonparametric bootstrap is nonparametric. That means it always does the right thing, except when it doesn't. It doesn't work when the sample size is too small or when the square root law doesn't hold or when the data are not IID or when various technical issues arise that are beyond the scope of this course – the parameter is not a nice enough function of the true unknown distribution.
- The parametric bootstrap is parametric. That means it is always wrong when the model is wrong. On the other hand, when the parametric bootstrap does the right thing (when the statistical model is correct), it does a much better job at smaller sample sizes than the nonparametric bootstrap.
- When the parameter θ is defined in terms of the parametric statistical model and can only be estimated using the parametric model (by maximum likelihood perhaps), the statistical model is needs to be correct for the parameter estimate $\hat{\theta}_n$ to make sense

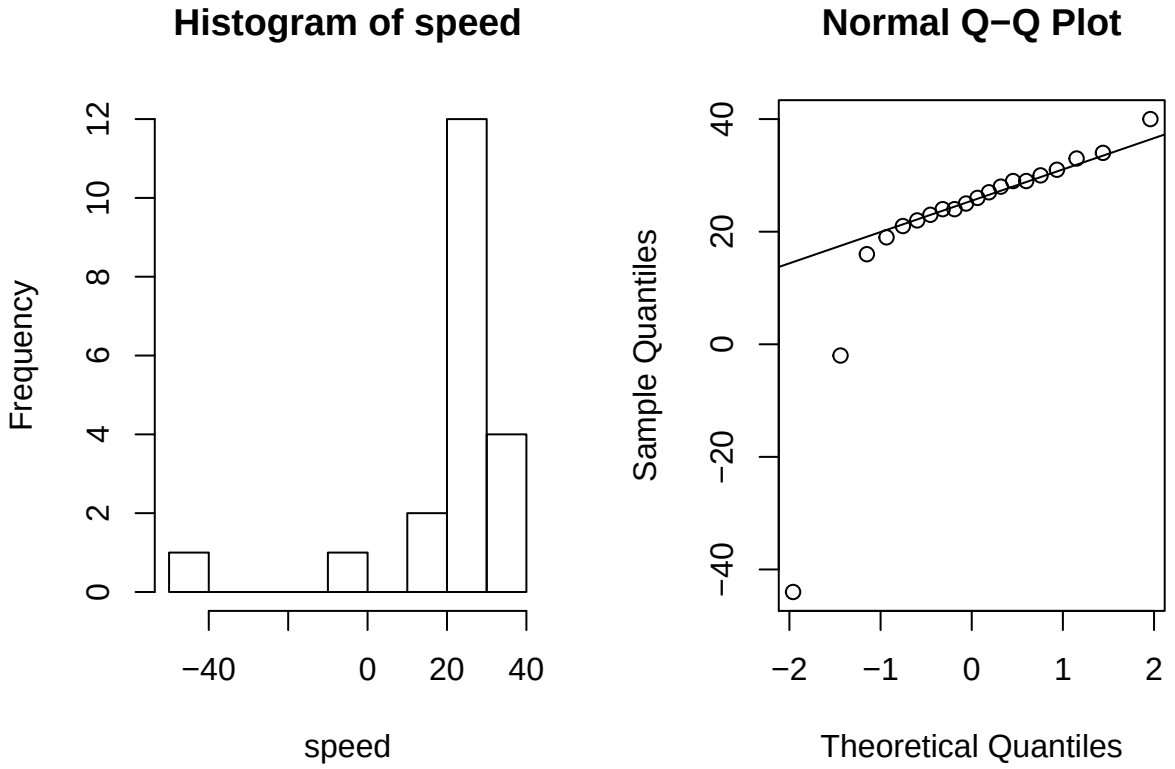
- Since we already need the statistical model to be correct, the parametric bootstrap is the logical choice

Abnormal speed of light data

- Thanks to Rob Gould (UCLA Statistics) for the following example
- In 1882 Simon Newcomb performed an experiment to measure the speed of light
- Measured time it took for light to travel from Fort Myer on the west bank of the Potomac River to a fixed mirror at the foot of the Washington monument 3721 meters away
- In the units of the data, the currently accepted "true" speed of light is 33.02
- Does the data support the current accepted speed of 33.02?

```
speed <- c(28, -44, 29, 30, 26, 27, 22, 23, 33, 16, 24, 29, 24, 40, 21, 31,
    34, -2, 25, 19)
```

- To convert these units to time in the millionths of a second, multiply by 10^{-3} and add 24.8



- A t-test assumes the population of measurements is normally distributed
- With this small sample size and a severe departure from normality, we can't be guaranteed a good approximation
- Instead, we can consider the bootstrap

1. State null and alternative hypotheses

$$H_0 : \mu = 33.02 \text{ versus } H_a : \mu \neq 33.02$$

2. Choose a significance level, in our case 0.05
3. Choose a test statistic, since we wish to estimate the mean speed we can use the sample average
4. Find the observed value of the test statistic
5. Calculate a p-value?

```
mean(speed)
```

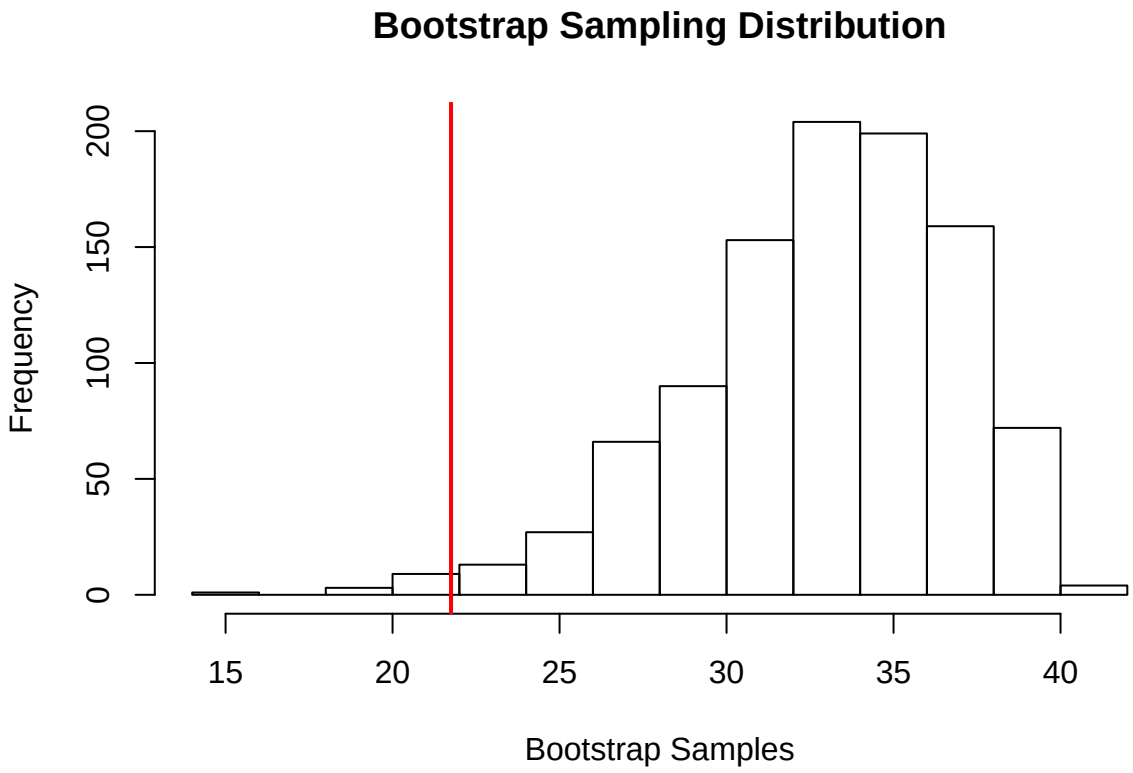
```
## [1] 21.75
```

- We now need a p-value, but we don't have the sampling distribution of our test statistic when the null hypothesis is true
- It is approximately normal, but that is a poor approximation here
- Instead we can perform a simulation under conditions in which we know the null hypothesis is true
- Use our data to represent the population, but first we shift it over so that the mean really is 33.02

```
newspeed <- speed - mean(speed) + 33.02
```

- Histogram of `newspeed` will have exactly the same shape as speed, but will be shifted
- Now we reach into our fake population and take out 20 observations at random, with replacement
- We take out 20 because that's the size of our initial sample
- We calculate the average and save it, then repeat this process many, many times
- Now we have a sampling distribution with mean 33.02
- Can compare this to our observed sample average and obtain a p-value

```
n <- 1000
bstrap <- double(n)
for (i in 1:n) {
    newsample <- sample(newspeed, 20, replace = T)
    bstrap[i] <- mean(newsample)
}
```



- Doesn't look normal, which means we did the right thing
- Not impossible for the sample average to be 21.75
- But it's not all that common, either
- The p-value is the probability of getting something more extreme than what we observed
- Notice 21.75 is $33.02 - 21.75 = 11.27$ units away from the null hypothesis

- So p-value is the probability of being more than 11.27 units away from 33.02

```
(sum(bstrap < 21.75) + sum(bstrap > 44.29))/1000
```

```
## [1] 0.011
```

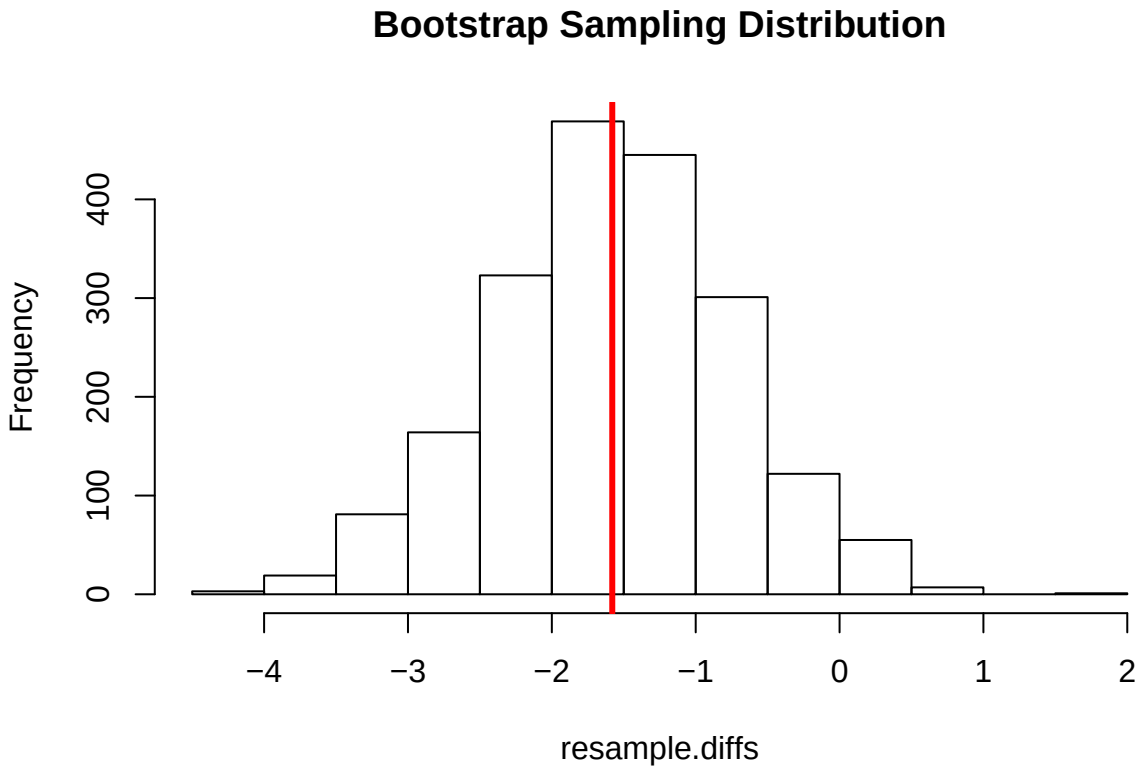
- Since our significance level is 5%, we reject H_0 and conclude that Newcomb's measurements were not consistent with the currently accepted figure

Example: Sleep study

- The two sample t-test checks for differences in means according to a known null distribution
- Similar to permutation tests
- Let's resample and generate the sampling distribution under the bootstrap assumption

```
bootstrap.resample <- function(object) sample(object, length(object), replace = TRUE)
diff.in.means <- function(df) {
    mean(df[df$group == 1, "extra"]) - mean(df[df$group == 2, "extra"])
}
resample.diffs <- replicate(2000, diff.in.means(sleep[bootstrap.resample(1:nrow(sleep)),
    ]))
```

```
hist(resample.diffs, main = "Bootstrap Sampling Distribution")
abline(v = diff.in.means(sleep), col = 2, lwd = 3)
```



Bootstrapping functions

- R has numerous built in bootstrapping functions, too many to mention, see `boot` library
- Example of the function `boot`
- Bootstrap of the **ratio of means** using the `city` data included in the `boot` package

```
library(boot)
data(city)
ratio <- function(d, w) sum(d$x * w)/sum(d$u * w)
results <- boot(city, ratio, R = 1000, stype = "w")
```

```
results
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = city, statistic = ratio, R = 1000, stype = "w")
##
##
## Bootstrap Statistics :
##     original     bias    std. error
## t1* 1.520313 0.04339365   0.2199604
```

```
boot.ci(results, type = "bca")
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = results, type = "bca")
##
## Intervals :
## Level       BCa
## 95%   ( 1.265,  2.092 )
## Calculations and Intervals on Original Scale
```

Bootstrapping a single statistic

- Can use the bootstrap to generate a 95% confidence interval for R-squared
- Linear regression of miles per gallon (`mpg`) on car weight (`wt`) and displacement (`disp`)
- Data source is `mtcars`
- The bootstrapped confidence interval is based on 1000 replications

```
rsq <- function(formula, data, indices) {
    d <- data[indices, ]
    fit <- lm(formula, data = d)
    return(summary(fit)$r.square)
}
results <- boot(data = mtcars, statistic = rsq, R = 1000, formula = mpg ~ wt +
    disp)
```

```
results
```

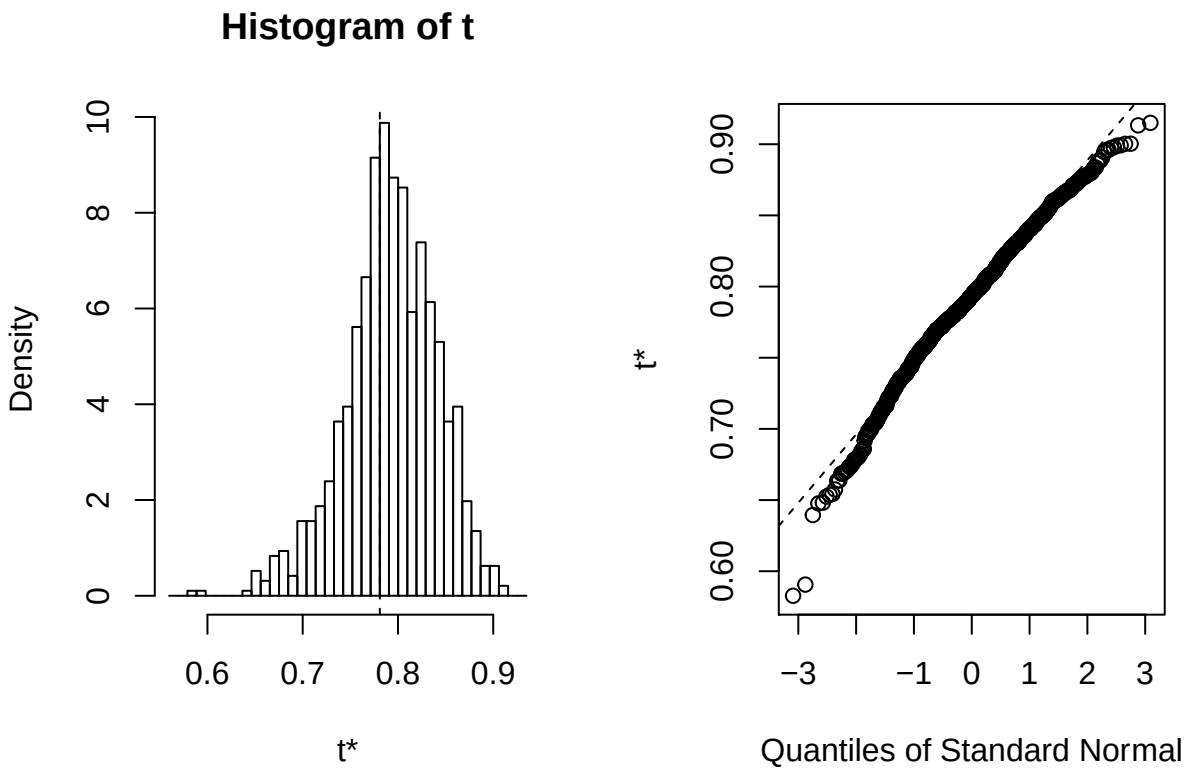
```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = mtcars, statistic = rsq, R = 1000, formula = mpg ~
```

```
##     wt + disp)
##
##
## Bootstrap Statistics :
##      original    bias    std. error
## t1* 0.7809306 0.0115427   0.0481346
```

```
boot.ci(results, type = "bca")
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = results, type = "bca")
##
## Intervals :
## Level       BCa
## 95%   ( 0.6500,  0.8574 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

```
plot(results)
```



Summary

- Bootstrapping provides a nonparametric approach to statistical inference when distributional assumptions may not be met
- Enables calculation of standard errors and confidence intervals in a variety of situations, e.g. medians, correlation coefficients, regression parameters, ...
- Hypothesis tests are a little more challenging

- The bootstrap is large sample, approximate, and asymptotic!
- Works when the empirical distribution $\hat{F}_n$ is close to the true unknown distribution F
- Usually the case when the sample size n is large and not otherwise, no method can save bad data!

Lecture 13: Cross Validation

Agenda

Suppose we have several different models for a particular data set. How should we choose the best one? Naturally, we would want to select the best performing model in order to choose the best. What is performance? How to estimate performance? Thinking about these ideas, we'll consider the following:

- Model assessment and selection
- Prediction error
- Cross-validation
- Smoothing example

Example: Lidar data

- Consider the `lidar` data from `SemiPar` package in R

```
library(SemiPar)
data(lidar)
```

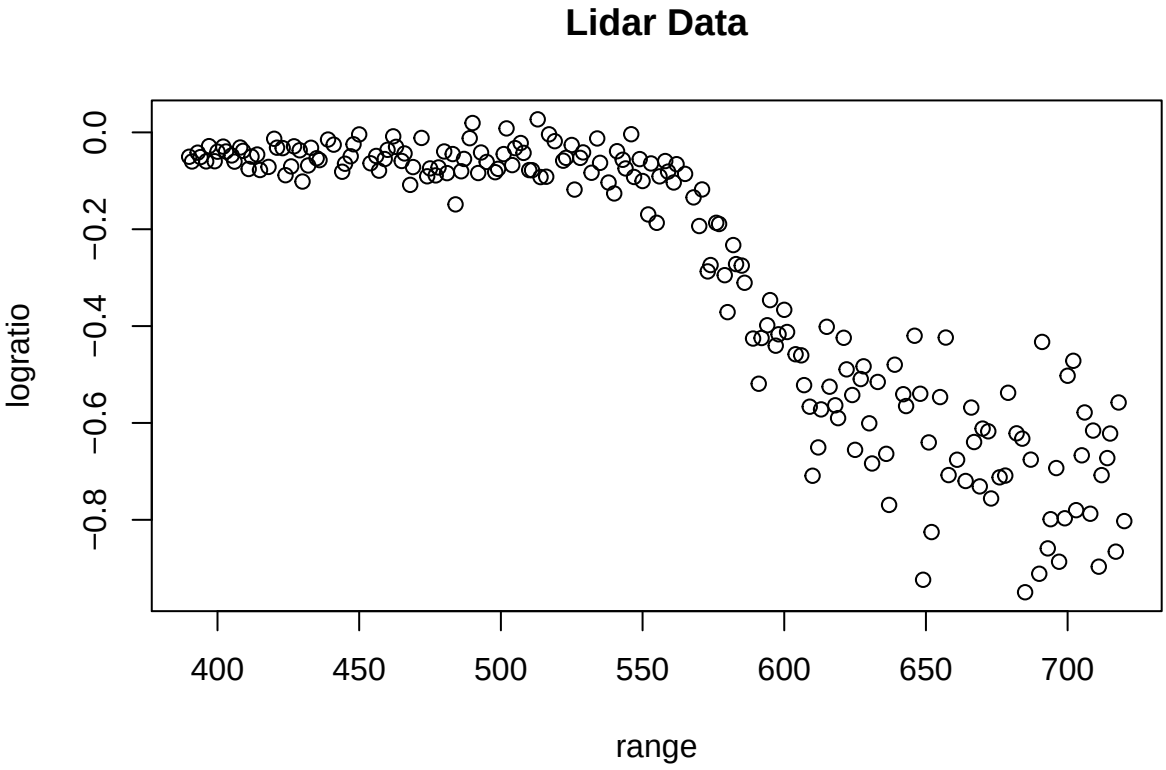
- Randomly split data into K subsets
- Each observation gets a `foldid` between `1` and `K`

```
K <- 10
n <- nrow(lidar)
foldid <- sample(rep(1:K, length = n))
```

- Split data into training and test data

```
lidar.train <- subset(lidar, foldid != 1)
lidar.test <- subset(lidar, foldid == 1)
attach(lidar.train)
```

```
plot(range, logratio, main = "Lidar Data")
```



loess (locally weighted smoothing)

- loess (locally weighted smoothing) is a popular tool that creates a smooth line through a timeplot or scatter plot to help you to see relationship between variables and foresee trends
- loess is typically to
  - Fitting a line to a scatter plot or time plot where noisy data values, sparse data points or weak interrelationships interfere with your ability to see a line of best fit
  - Linear regression where least squares fitting doesn't create a line of good fit or is too labor-intensive to use
  - Data exploration and analysis

Nonparametric smoothing

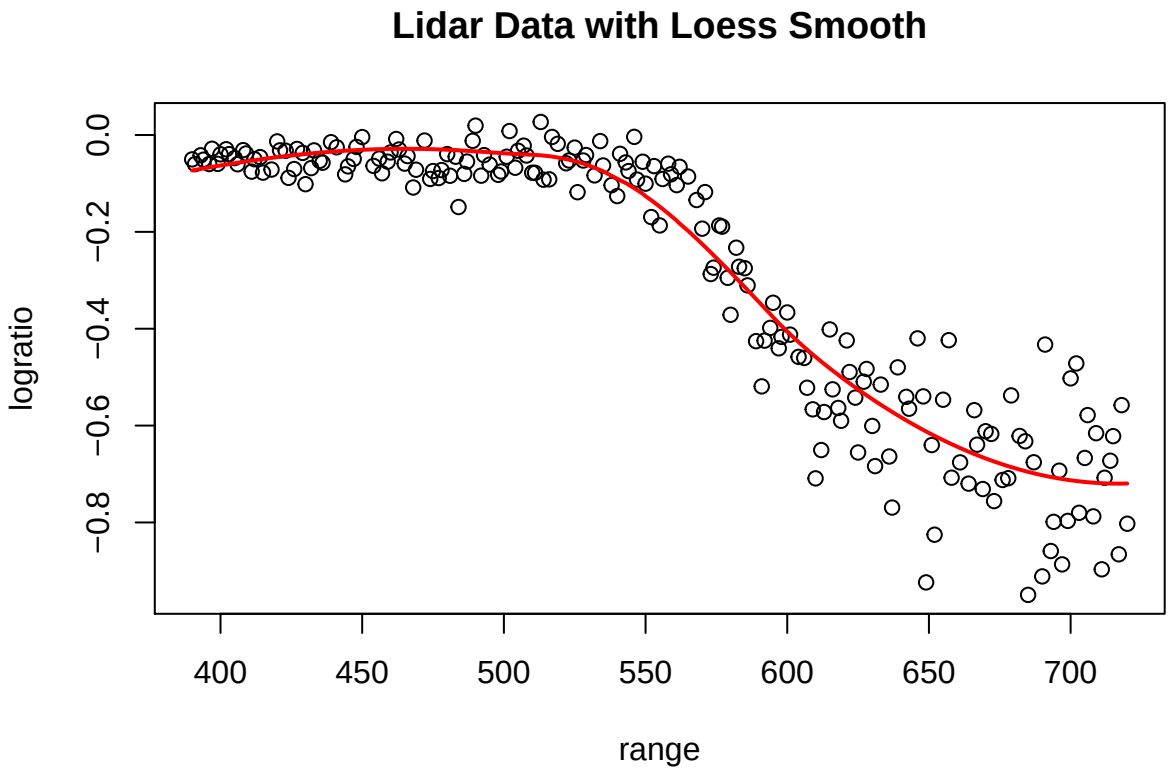
- Benefits
  - Provides a flexible approach to representing data
  - Ease of use
  - Computations are relatively easy (sometimes)
- Disadvantages
  - No simple equation for a set of data
  - Less understood than parametric smoothers
  - Depends on a `span` parameter controlling the smoothness

Example: Lidar data

- Can consider fitting a `loess` smooth to the data

```
obj0 <- loess(logratio ~ range, data = lidar.train, control = loess.control(surface = "direct"))
```

```
plot(obj0, xlab = "range", ylab = "logratio", main = "Lidar Data with Loess Smooth")
points(obj0$x, obj0$fitted, type = "l", col = "red", lwd = 2)
```



- How to choose the **span** parameter?
- Which model is the best? Why?

```
obj3 <- loess(logratio ~ range, data = lidar.train, span = 1, control = loess.control(surface = "direct
obj2 <- loess(logratio ~ range, data = lidar.train, span = 0.3, control = loess.control(surface = "dire
obj1 <- loess(logratio ~ range, data = lidar.train, span = 0.02, control = loess.control(surface = "dir
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : span too small. fewer data values than degrees of freedom.
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : pseudoinverse used at 390
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : neighborhood radius 3
```

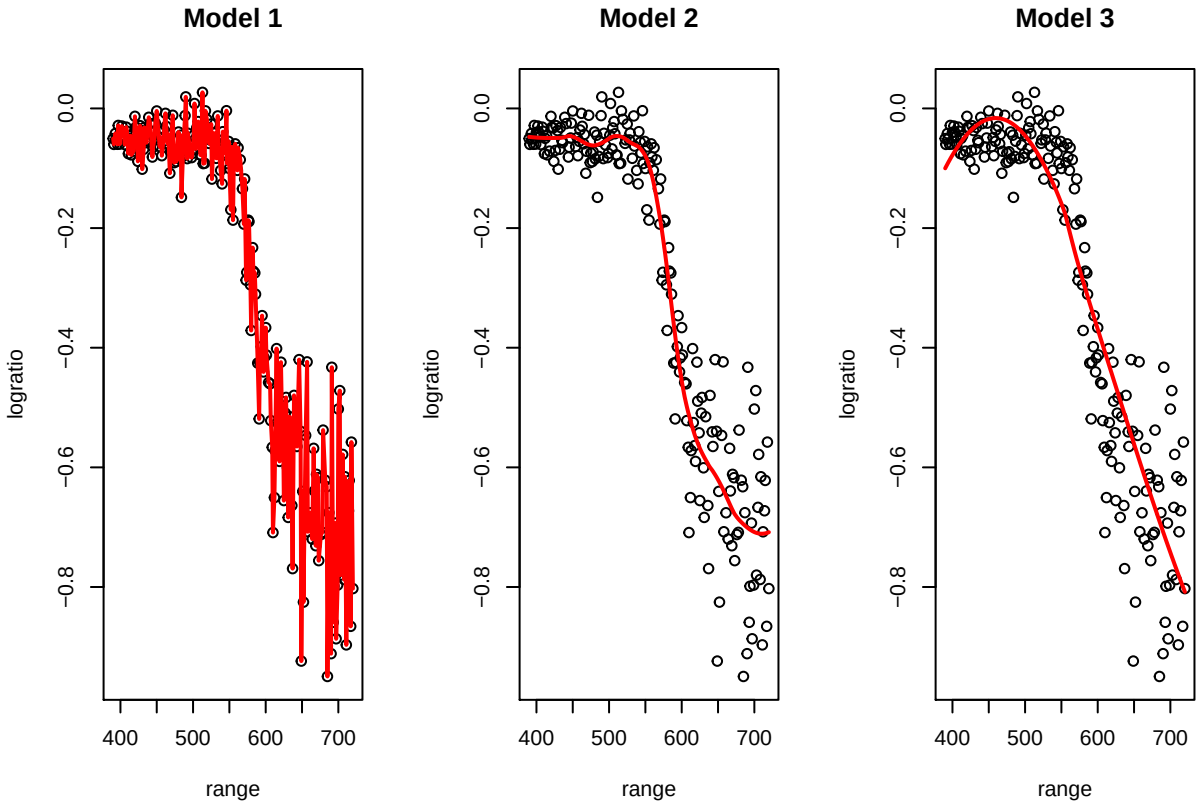
```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : reciprocal condition number 0
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : There are other near singularities as well. 4
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 198
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 198
```

```
## Warning in sqrt(sum.squares/one.delta): NaNs produced
```



- Model 1: $\sum_i \left|Y_i - \hat{f}(X_i)\right|^2 = 0$
- Model 2: $\sum_i \left|Y_i - \hat{f}(X_i)\right|^2 = 1.03$
- Model 3: $\sum_i \left|Y_i - \hat{f}(X_i)\right|^2 = 1.71$

Therefore, Model 1 has the smallest squared error over the data

```
c(sum((logratio - obj1$fitted)^2), sum((logratio - obj2$fitted)^2), sum((logratio -
    obj3$fitted)^2))
```

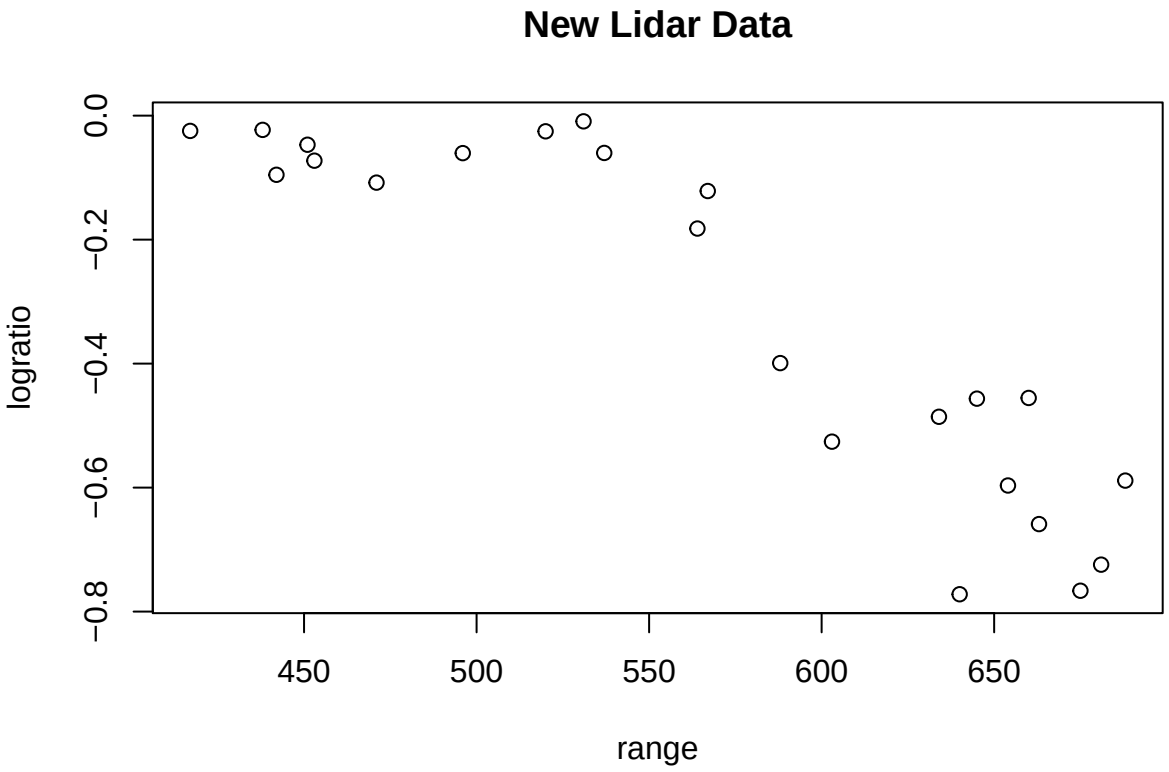
```
## [1] 1.483014e-30 1.183769e+00 1.808425e+00
```

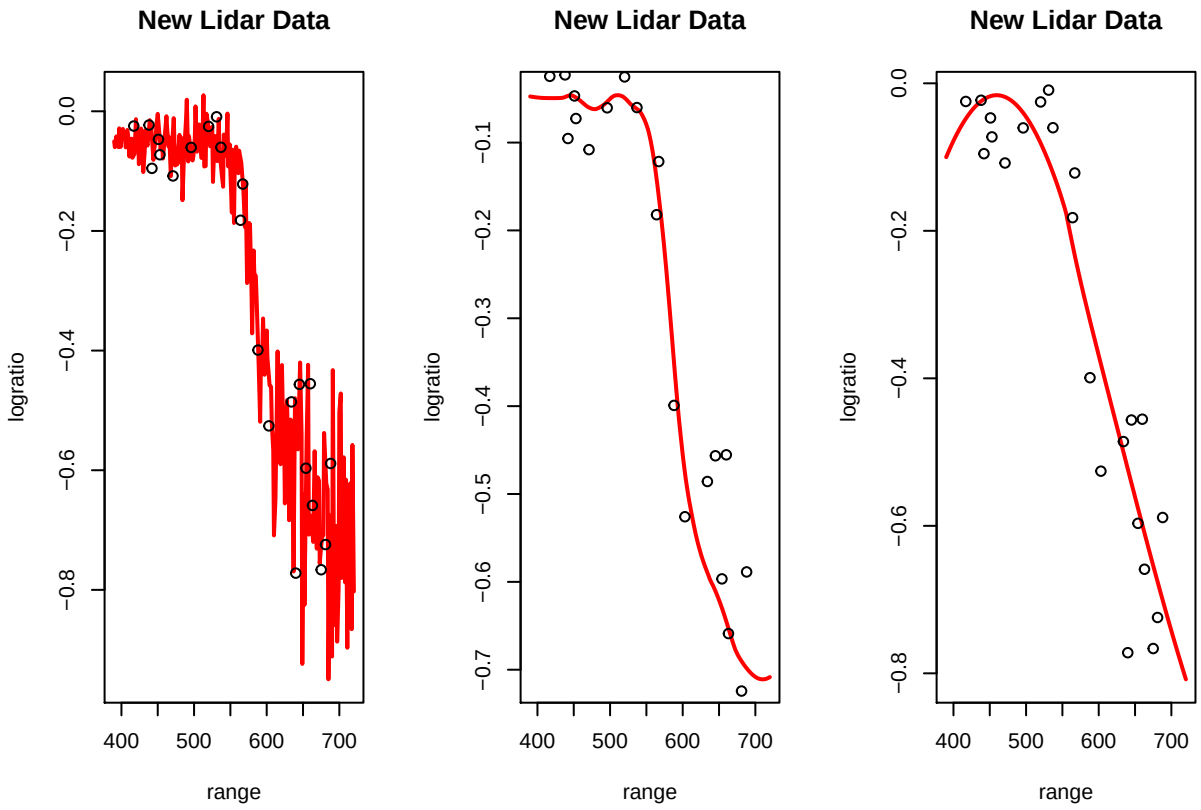
```
c(mean((logratio - obj1$fitted)^2), mean((logratio - obj2$fitted)^2), mean((logratio -
    obj3$fitted)^2))
```

```
## [1] 7.489972e-33 5.978629e-03 9.133459e-03
```

- Suppose now that we have new data from a second experiment, `lidar.test`
- How well do my three models predict the new data? Consider again $\sum_i \left|Y_i^{(new)} - \hat{f}(X_i^{(new)})\right|^2$

```
plot(lidar.test$range, lidar.test$logratio, xlab = "range", ylab = "logratio",
    main = "New Lidar Data")
```






```
y.hat <- predict(obj1, newdata = lidar.test)
sum((lidar.test$logratio - y.hat)^2)
mean((lidar.test$logratio - y.hat)^2)
```

```
y.hat <- predict(obj2, newdata = lidar.test)
sum((lidar.test$logratio - y.hat)^2)
mean((lidar.test$logratio - y.hat)^2)

y.hat <- predict(obj3, newdata = lidar.test)
sum((lidar.test$logratio - y.hat)^2)
mean((lidar.test$logratio - y.hat)^2)
```

- Model 1: $\sum_i \left|Y_i^{(new)} - \hat{f}(X_i^{(new)})\right|^2 = 1.23$
- Model 2: $\sum_i \left|Y_i^{(new)} - \hat{f}(X_i^{(new)})\right|^2 = 0.17$
- Model 3: $\sum_i \left|Y_i^{(new)} - \hat{f}(X_i^{(new)})\right|^2 = 0.24$

Now notice Model 2 has the smallest squared error over the **new** data

We can also consider average squared error since the original and new data contain a different number of points. That is,

$$\frac{1}{n}\sum_{i=1}^{n}\left|Y_i - \hat{f}(X_i)\right|^2 ,$$

computed using (X, Y) belonging to either the original or new data.

ASE on	Model 1	Model 2	Model 3
Fitting Data	0	0.0058	0.0091
New Data	0.0536	0.0075	0.0102

Model Assessment and Selection

Basic problem: We have several different models so how can we choose the best one? We could estimate the performance of each model in order to choose the best one. What is performance? How to estimate performance?

- **Model assessment**: Evaluating how closely a particular model fits the data
- **Model selection**: Choosing the best model among several different ones

Model Assessment and Selection

Model selection is ubiquitous. Where do we use model selection?

- Linear regression (variable selection)
- Smoothing (smoothing parameter selection)
- Kernel density estimation (bandwidth selection)
- ...

Prediction Error

- Prediction error is one measure of performance of a model
- In short, we're interested in how well the model make predictions about new data?

- Exact definition (e.g. squared error or other) depends on problem
- Consider either a nonparametric (smoothing) or linear regression modeling setting

- In this case, the regression equation is fit using observed data

$$(X_1, Y_1), \ldots, (X_n, Y_n) \ ,$$

  resulting in a regression function $\hat{f}_n(\cdot)$
- The resulting model can be used for prediction at new X values resulting in $\hat{f}_n(X_{n+1})$
- One measure of prediction error for regression is the Mean Squared Prediction Error (MSPE), i.e.

$$\text{MSPE} = \text{E}\left|Y_{n+1} - \hat{f}_n(X_{n+1})\right|^2$$

Question: How do we estimate prediction error if we only have n observations?

Answer: If we had m additional independent observations $(X_{n+1}, Y_{n+1}), \ldots, (X_{n+m}, Y_{n+m})$, then we could estimate the MSPE by

$$\frac{1}{m}\sum_{i=1}^{m}\left|Y_{n+i} - \hat{f}_n(X_{n+i})\right|^2$$

Notice, if we **reuse** the same data to estimate MPSE we have an in-sample estimate of the MPSE

$$\frac{1}{n}\sum_{i=1}^{n}\left|Y_{i} - \hat{f}_n(X_{i})\right|^2 \ .$$

But, as we saw in the `lidar` example this is generally a bad (overly optimistic) estimate of the MSPE. Using the same data for fitting and estimating prediction error generally leads to bad estimates of prediction error that are overly optimistic. This is related to the phenomenon known as **overfitting**.

How do we estimate prediction error if we only have n observations?

Cross-Validation

- Cross-validation is a method for estimating prediction error
- Main idea is to split the data into two parts
  - **training** portion used for fitting the model
  - **test** portion for validating the model or estimating the prediction error

K-fold Cross-Validation

1. Split the data into K roughly equal-sized parts
2. For the kth part
   - Fit the model using the remaining $K-1$ parts
   - Calculate the prediction error of the fitted model when predicting for the kth part of the data

Example: $K = 5$

Fold 1	Validation	Train	Train	Train	Train
Fold 2	Train	Validation	Train	Train	Train
Fold 3	Train	Train	Validation	Train	Train
Fold 4	Train	Train	Train	Validation	Train
Fold 5	Train	Train	Train	Train	Validation

Leave-one-out cross validation

- A special case of cross-validation is known as **leave-one-out cross validation**
- Simply K-fold cross-validation where $K = n$

1. For the kth observation
   - Fit the model using the remaining $k - 1$ observations
   - Calculate the prediction error of the fitted model when predicting for the kth observation

Pseudo-code

The following is some pseudo-code for K-fold cross-validation.

```
K <- 10
n <- nrow(mydata)
cv.error <- vector(length = K)

# Randomly split data into K subsets â€" each observation gets a foldid
# between 1 and K
foldid <- sample(rep(1:K, length = n))

# Repeat K times
for (i in 1:K) {
    # Fit using training set
    f.hat <- estimator(mydata[foldid != i, ])

    # Calculate prediction error on validation set
    cv.error[i] <- calc_error(f.hat, mydata[foldid == i, ])
}

cv.error.estimate <- mean(cv.error)
```

Example: Lidar Data

Using the `lidar` data again, we will

1. Fit several different `loess` smooths corresponding to different choices of bandwidth (span)
2. Estimate the mean squared prediction error of each smooth
3. Choose the smooth with the smallest estimated MSPE

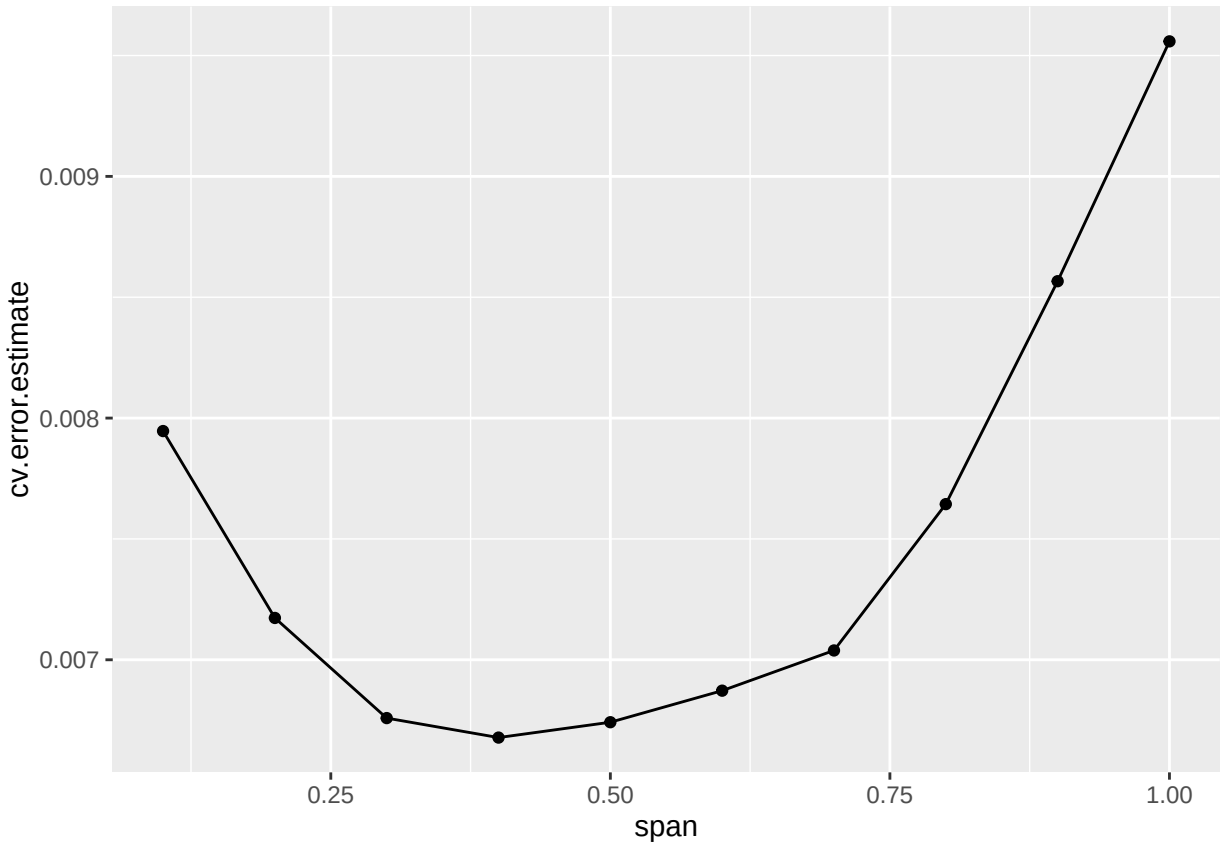
```
library(ggplot2)
library(SemiPar)
data(lidar)
s = seq(from = 0.1, to = 1, by = 0.1)
K <- 10
n <- nrow(lidar)
```

```
cv.error <- matrix(nrow = K, ncol = length(s))
foldid <- sample(rep(1:K, length = n))

for (i in 1:K) {
    # Fit, predict, and calculate error for each bandwidth
    cv.error[i, ] <- sapply(s, function(span) {
        # Fit LOESS model to training set
```

```
        obj <- loess(logratio ~ range, data = subset(lidar, foldid != i), span = span,
            control = loess.control(surface = "direct"))
        # Predict and calculate error on the validation set
        y.hat <- predict(obj, newdata = subset(lidar, foldid == i))
        pse <- mean((subset(lidar, foldid == i)$logratio - y.hat)^2)
        return(pse)
    })
}

# Columns of cv.error correspond to different bandwidths
cv.error.estimate <- colMeans(cv.error)
```

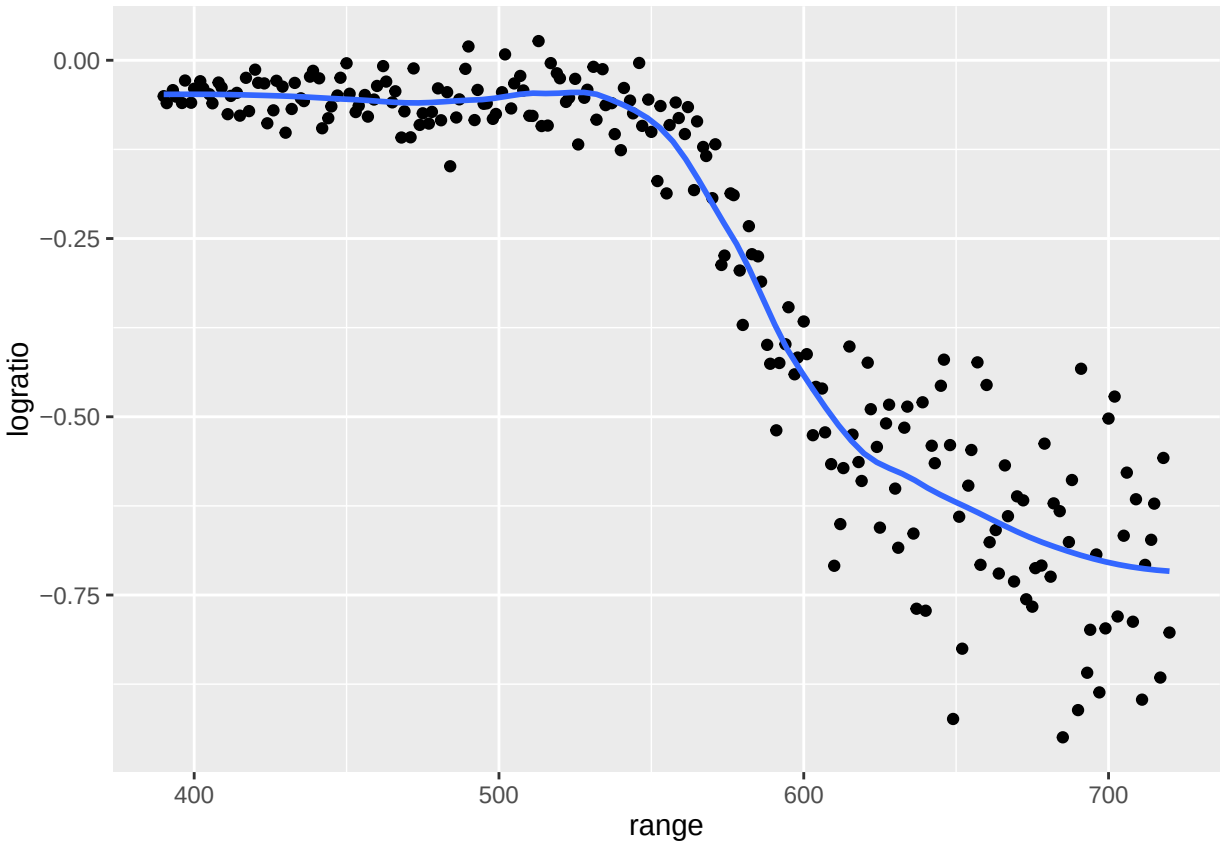
Cross-validation estimates of MSPE

```
qplot(s, cv.error.estimate, geom = c("line", "point"), xlab = "span")
```



Smoother selected by cross-validation

```
s.best <- s[which.min(cv.error.estimate)]
qplot(range, logratio, data = lidar) + geom_smooth(method = "loess", span = s.best,
    se = FALSE)
```



Summary

- Can assess models by prediction error
- Select model with smallest prediction error
- Using same data for fitting and estimating prediction error leads to overfitting
- Cross-validation is a method for estimating prediction error that avoids overfitting

Lecture 14: Density Estimation

Agenda

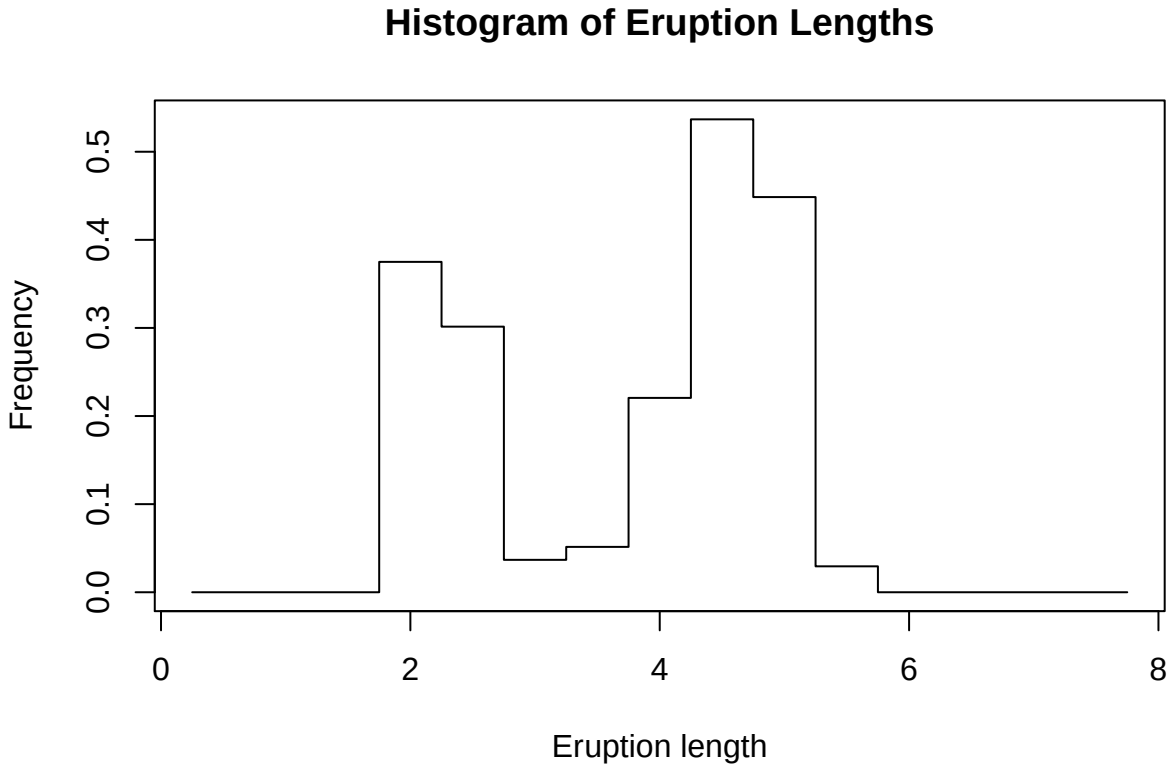
- Histograms
- Glivenko-Cantelli theorem
- Error for density estimates
- Kernel density estimates
- Bivariate density estimates

Histograms

- Histograms are one of the first things learned in "Introduction to Statistics"
- Simple way of estimating a distribution
  - Split the sample space up into bins
  - Count how many samples fall into each bin
- If we hold the bins fixed and take more and more data, then the relative frequency for each bin will converge on the bin's probability

Example: Old Faithful Geyser Data

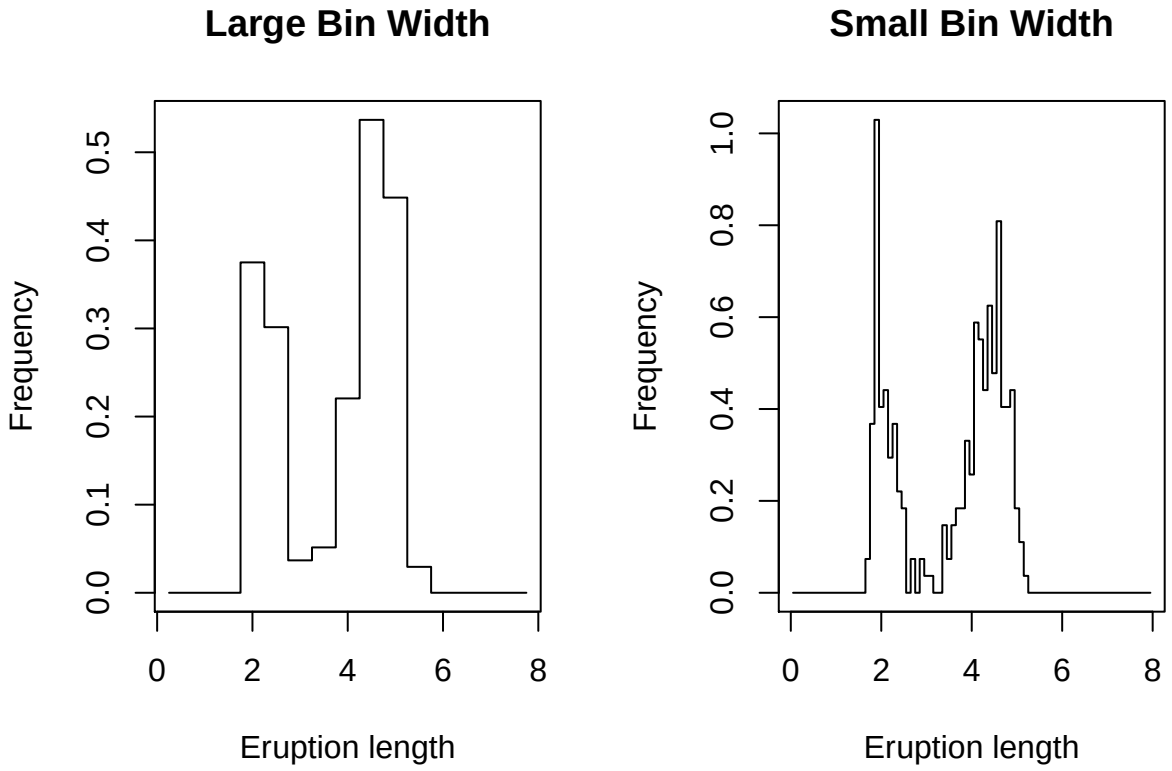
```
data(faithful)
x0 <- 0
x1 <- 8
h <- 0.5
my.breaks <- seq(from = x0, to = x1, by = h)
myhist <- hist(faithful$eruptions, breaks = my.breaks, right = F, plot = F)
plot(myhist$mids, myhist$density, type = "s", xlab = "Eruption length", ylab = "Frequency",
    main = "Histogram of Eruption Lengths")
```



Histograms

- What about a density function?
- Could take our histogram estimate and say that the probability density is uniform within each bin
- Gives us a piecewise-constant estimate of the density
- ***Problem***: Will not converge on the true density – unless we shrink the bins as we get more and more data
- Bias-variance trade-off
  - A large number of very small bins, the minimum bias in our estimate of any density becomes small
  - But the variance grows for very small bins

Example: Old Faithful Geyser Data



Histograms

Bin width primarily controls the amount of smoothing, lots of guidance available

1. **Sturges' rule**: Optimal width of class intervals is given by

$$\frac{R}{1+\log_2 n}$$

   where R is the sample range – Designed for data sampled from symmetric, unimodal populations
2. **Scott's Normal reference rule**: Specifies a bin width

$$3.49\hat{\sigma}n^{-1/3}$$

   where $\hat{\sigma}$ is an estimate of the population standard deviation σ
3. **Freedman-Diaconis rule**: Specifies the bin width to be

$$2(IQR)n^{-1/3}$$

   where the IQR is the sample inter-quartile range

- Is learning the whole distribution non-parametrically even feasible?
- How can we measure error to deal with the bias-variance trade-off?

Empirical CDF

- Learning the whole distribution is ***feasible***
- Something even dumber than shrinking histograms will work

- Suppose we have one-dimensional samples $x_1, \ldots, x_n$ with CDF F
- Define the empirical cumulative distribution function on n samples as

$$\hat{F}_n(a) = \frac{1}{n} \sum_{i=1}^{n} I(-\infty < x_i \leq a)$$

- Just the fraction of the samples which are less than or equal to a

Glivenko-Cantelli theorem

- Then the ***Glivenko-Cantelli theorem*** says

$$\max_a |\hat{F}_n(a) - F(a)| \to 0$$

- So the empirical CDF converges to the true CDF ***everywhere***, i.e. the maximum gap between the two of them goes to zero
- Pitman (1979) calls this the "fundamental theorem of statistics"
- Can learn distributions just by collecting enough data
- Can we use the empirical CDF to estimate a density?
- Yes, but it's discrete and doesn't estimate a density well
- Usually we can expect to find some new samples between our old ones
- So we want a non-zero density between our observations
- Uniform distribution within each bin of a histogram doesn't have this issue
- Can we do better?

Error for density estimates

- Yes, but what do we mean by "better" density estimates?
- Three ideas:
  1. Squared deviation from the true density should be small

$$\int \left(f(x) - \hat{f}(x)\right)^2 dx$$

  2. Absolute deviation from the true density should be small

$$\int \left|f(x) - \hat{f}(x)\right| dx$$

  3. Average log-likelihood ratio should be kept low

$$\int f(x) \log \frac{f(x)}{\hat{f}(x)} dx$$

- Squared deviation is similar to MSE criterion used in regression
  - Used most frequently since it's mathematically tractable
- Absolute deviation considers L_1 or total variation distance between the true and the estimated density
  - Nice property that $\frac{1}{2} \int \left|f(x) - \hat{f}(x)\right| dx$ is exactly the maximum error in our estimate of the probability of any set
  - But it's tricky to work with, so we'll skip it
- Minimizing the log-likelihood ratio is intimately connected both to maximizing the likelihood and to minimizing entropy
  - Called Kullback-Leibler divergence or relative entropy

- Notice that
$$\int \left(f(x) - \hat{f}(x)\right)^2 dx = \int f^2(x)dx - 2\int f(x)\hat{f}(x)dx + \int \hat{f}^2(x)dx$$
- First term doesn't depend on the estimate, so we can ignore it for purposes of optimization
- Third term only involves $\hat{f}(x)$, and is just an integral, which we can do numerically
- Second term involves both the true and the estimated density; we can approximate it using Monte Carlo by
$$\frac{2}{n}\sum_{i=1}^{n}\hat{f}(x_i)$$
- Then our error measure is
$$\frac{2}{n}\sum_{i=1}^{n}\hat{f}(x_i) + \int \hat{f}^2(x)dx$$
- In fact, this error measure does not depend on having one-dimension data
- For purposes of cross-validation, we can estimate $\hat{f}(x)$ on the training set and then restrict the sum to points in the testing set

Naive estimator

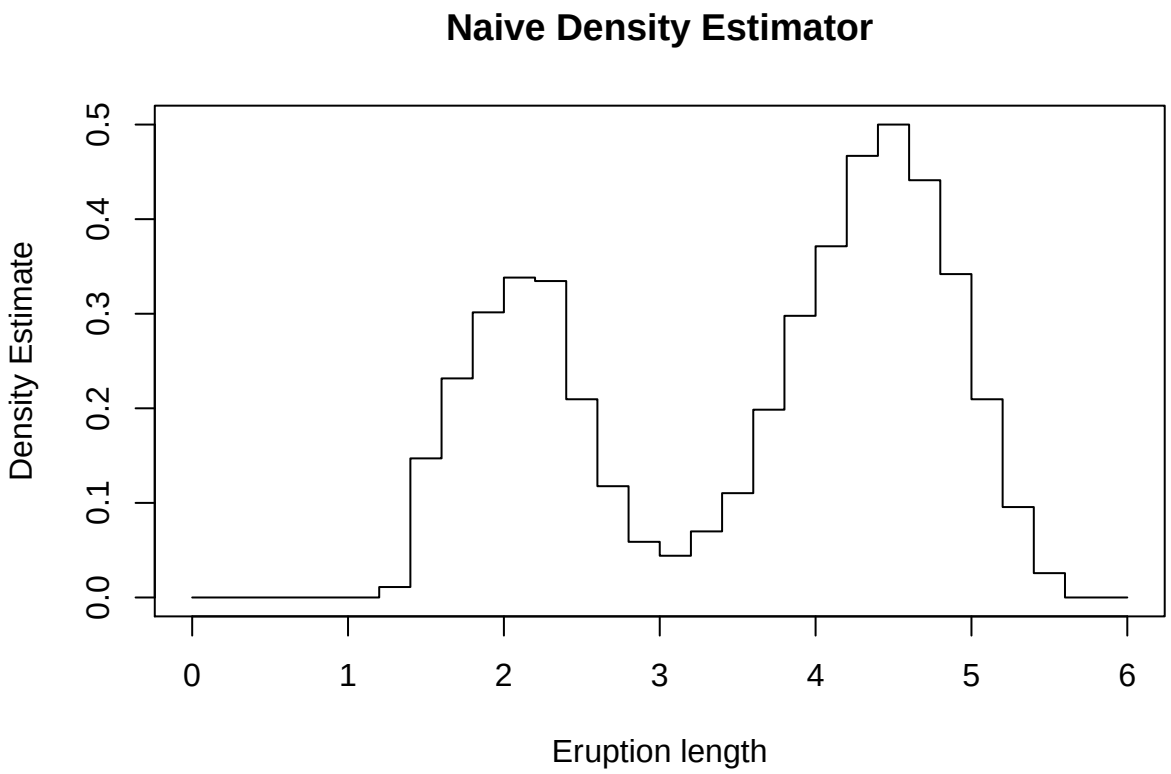
- If a random variable X has probability density f, then
$$f(x) = \lim_{h\to 0}\frac{1}{2h}P(x-h < X < x+h)$$
- Thus, a naive estimator would be
$$\widehat{f}(x) = \frac{1}{2nh}\left[\#\text{ of } x_i \text{ falling in } (x-h, x+h)\right]$$
- Or, equivalently
$$\widehat{f}(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}w\left(\frac{x-x_i}{h}\right)$$
where w is a weight function defined as
$$w(x) = \begin{cases} 1/2 & |x| < 1 \\ 0 & \text{otherwise} \end{cases}$$
- In short, a naive estimate is constructed by placing a box of width $2h$ and height $\frac{1}{2nh}$ on each observation, then summing to obtain the estimate

Example: Old Faithful Geyser Data

```
my.w <- function(x) {
    if (abs(x) < 1)
        w <- 1/2 else w <- 0
    w
}
x <- seq(0, 6, 0.2)
m <- length(x)
n <- length(faithful$eruptions)
h <- 0.5
fhat <- rep(0, m)
```

```
for (i in 1:m) {
    S <- 0
    for (j in 1:n) {
        S <- S + (1/h) * my.w((faithful$eruptions[j] - x[i])/h)
    }
    fhat[i] <- (1/n) * S
}
```

```
plot(x, fhat, type = "s", xlab = "Eruption length", ylab = "Density Estimate",
    main = "Naive Density Estimator")
```



Naive estimator

- Not wholly satisfactory, from the point of view of using density estimates for presentation
- Estimate $\widehat{f}$ is a step function
- In the formula for the naive estimate, we can replace the weight function w by another function K with more desirable properties
- Function K is called a **kernel**

Kernel density estimates

- Resulting estimate is a **kernel estimator**:

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}K\left(\frac{x - x_i}{h}\right).$$

- h is the **window width**, **smoothing parameter**, or **bandwidth**
- Usually the kernel K is taken to be a probability density function itself (i.e., normal density)

- Resulting estimate will inherit all the smoothness properties of K

Most popular choices for the kernel K are

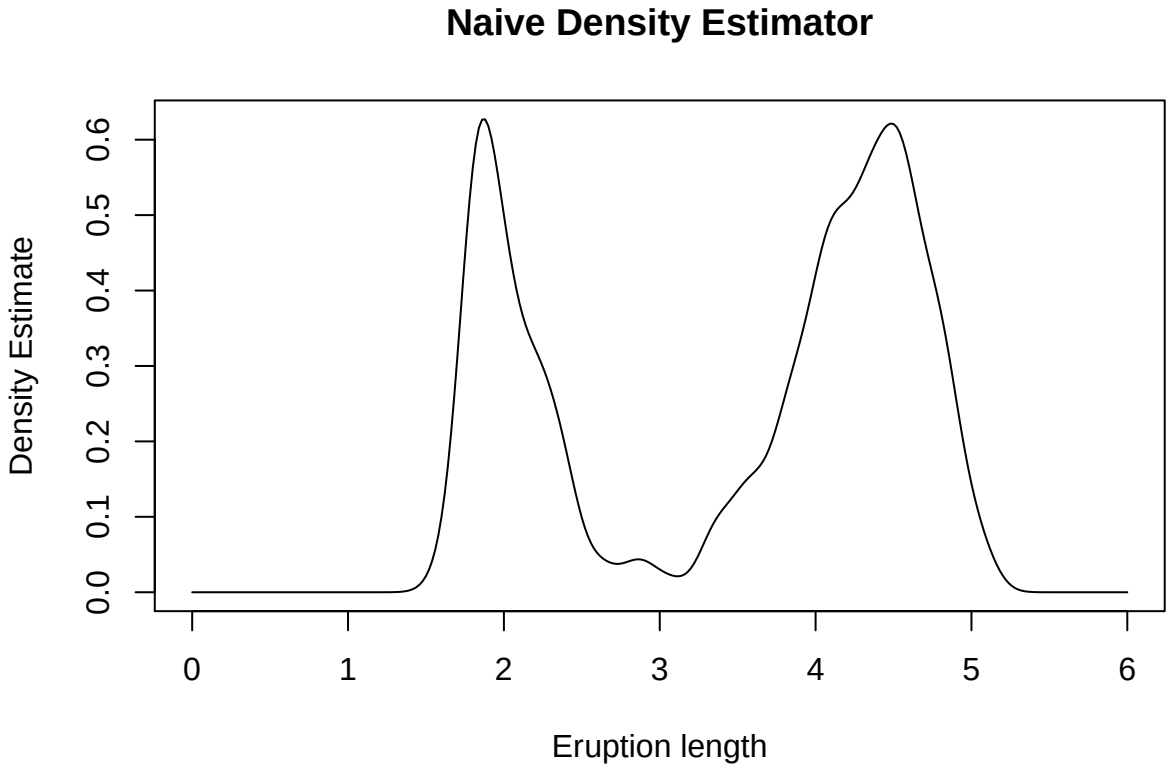
Family	Kernel
Gaussian	$K(t) = \frac{1}{\sqrt{2\pi}} e^{-t^2/2}$
Rectangular	$K(t) = 1/2$ for $
Triangular	$K(t) = 1-
Epanechnikov	$K(t) = \frac{3}{4}(1 - (1/5)t^2)$ for $

```
my.w <- function(x, type = "gaussian") {
    if (type == "gaussian") {
        w <- dnorm(x)
        return(w)
    }
    if (type == "naive") {
        if (abs(x) < 1)
            w <- 1/2 else w <- 0
        return(w)
    }
    print("You have asked for an undefined kernel.")
    return(NULL)
}
```

Example: Old Faithful Geyser Data

```
x <- seq(0, 6, 0.02)
m <- length(x)
n <- length(faithful$eruptions)
h <- 0.1
fhat <- rep(0, m)
for (i in 1:m) {
    S <- 0
    for (j in 1:n) {
        S <- S + (1/h) * my.w((faithful$eruptions[j] - x[i])/h)
    }
    fhat[i] <- (1/n) * S
}
```

```
plot(x, fhat, type = "l", xlab = "Eruption length", ylab = "Density Estimate",
    main = "Naive Density Estimator")
```



Bandwidth selection

- Cross-validation, which could be time consuming
- Optimal bandwidth for a Gaussian kernel to estimate a Gaussian distribution is $1.06\sigma/n^{1/5}$
- Called the **Gaussian reference rule** or the **rule-of-thumb** bandwidth
- When you call `density` in R, this is basically what it does

Kernel density estimate samples

- There are times when one wants to draw a random sample from the estimated distribution
- Easy with kernel density estimates, because each kernel is itself a probability density
- Suppose the kernel is Gaussian, that we have scalar observations $x_1, \ldots, x_n$ and bandwidth h
  1. Pick an integer uniformly at random from 1 to n
  2. Use `rnorm(1,x[i],h)`, or `rnorm(q,sample(x,q,replace=TRUE),h)` for q draws
- Using a different kernel, we'd just need to use the random number generator function for the corresponding distribution

Other Approaches

- Histograms and kernels are not the only possible way of estimating densities
- Can try the local polynomial trick, series expansions, splines, penalized likelihood approaches, etc
- For some of these, avoid negative probability density estimates using the log density

Density estimation in R

- `density()` function is the most common

```
density(x, ...)
## Default S3 method:
density(x, bw = "nrd0", adjust = 1, kernel = c("gaussian", "epanechnikov", "rectangular",
    "triangular", "biweight", "cosine", "optcosine"), weights = NULL, window = kernel,
    width, give.Rkern = FALSE, n = 512, from, to, cut = 3, na.rm = FALSE, ...)
```

- `ASH` and `KernSmooth` are both fast, accurate, and well-maintained (Deng and Wickham, 2011)

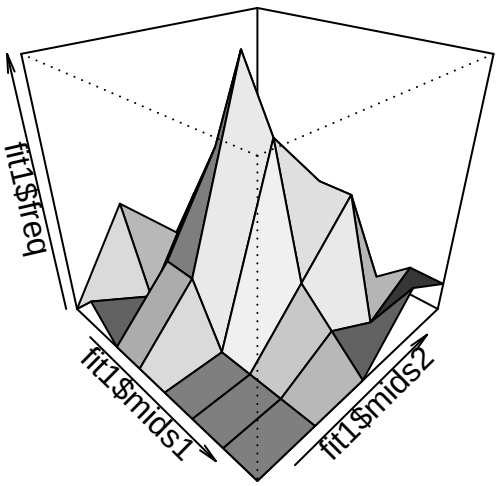
Bivariate density estimation

- To construct a bivariate density histogram, it is necessary to define two-dimensional bins and count the number of observations in each bin
- Can use `bin2d` function in R will bin a bivariate data set

```
bin2d <- function(x, breaks1 = "Sturges", breaks2 = "Sturges") {
    histg1 <- hist(x[, 1], breaks = breaks1, plot = FALSE)
    histg2 <- hist(x[, 2], breaks = breaks2, plot = FALSE)
    brx <- histg1$breaks
    bry <- histg2$breaks
    freq <- table(cut(x[, 1], brx), cut(x[, 2], bry))

    return(list(call = match.call(), freq = freq, breaks1 = brx, breaks2 = bry,
        mids1 = histg1$mids, mids2 = histg2$mids))
}
```

- Following example computes the bivariate frequency table
- After binning the data, the `persp` function plots the density histogram

```
data(iris)
fit1 = bin2d(iris[1:50, 1:2])
persp(x = fit1$mids1, y = fit1$mids2, z = fit1$freq, shade = T, theta = 45,
    phi = 30, ltheta = 60)
```



Bivariate kernel methods

- Suppose the data is $X_1, \ldots, X_n$, where each $X_i \in \mathbb{R}^2$
- Kernel density estimates can be extended to a multivariate (bivariate) setting

- Let $K(\cdot)$ be a bivariate kernel (typically a bivariate density function), then bivariate kernel density estimate is

$$\hat{f}(X) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{X - X_i}{h}\right)$$

Example: Bivariate normal

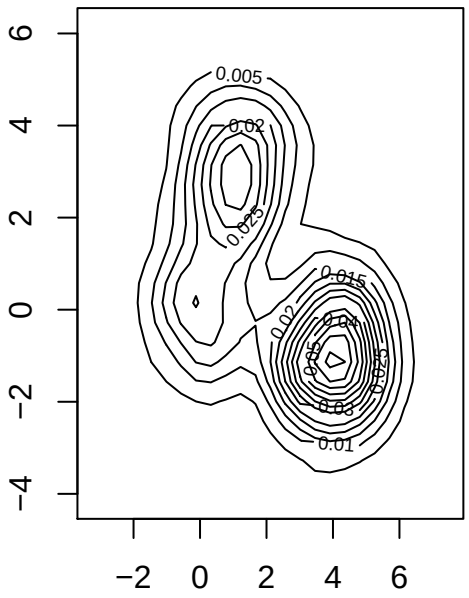
- Estimate the bivariate density when the data is generated from a mixture model with three components with identical covariance $\Sigma = I_2$ and different means

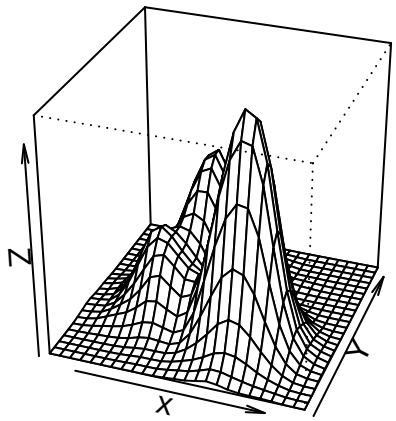
$$\mu_1 = (0, 0) \quad \mu_2 = (1, 3), \quad \mu_3 = (4, -1).$$

- Mixture probabilities are $p = (0.2, 0.3, 0.5)$

```
library(MASS)
n <- 2000
p <- c(0.2, 0.3, 0.5)
mu <- matrix(c(0, 1, 4, 0, 3, -1), 3, 2)
Sigma <- diag(2)
i <- sample(1:3, replace = TRUE, prob = p, size = n)
k <- table(i)

x1 <- mvrnorm(k[1], mu = mu[1, ], Sigma)
x2 <- mvrnorm(k[2], mu = mu[2, ], Sigma)
x3 <- mvrnorm(k[3], mu = mu[3, ], Sigma)
X <- rbind(x1, x2, x3)  #the mixture data
x <- X[, 1]
y <- X[, 2]

fhat <- kde2d(x, y, h = c(1.87, 1.84))
```

```
par(mfrow = c(1, 2))
contour(fhat)
persp(fhat, phi = 30, theta = 20, d = 5, xlab = "x")
```





```r
par(mfrow = c(1, 1))
```

Summary

- Over 20 packages that perform density estimation (Deng and Wickham, 2011)
- Kernel density estimation is the most common approach
- Density estimation can be parametric, where the data is from a known family
- Bayesian approaches are also available

Lecture 15: Bayesian Statistics

Agenda

- Bayesian inference
- Priors
- Point estimates
- Bayesian hypothesis testing
- Bayes factors

Bayesian Inference

- Everything we have done up to now is frequentist statistics, Bayesian statistics is very different
  - Bayesians don't do confidence intervals and hypothesis tests
  - Bayesians don't use sampling distributions of estimators
  - Modern Bayesians aren't even interested in point estimators
- So what do they do? Bayesians treat parameters as random variables

To a Bayesian probability is the only way to describe uncertainty. Things not known for certain – like values of parameters – must be described by a probability distribution

- Suppose you are uncertain about something, which is described by a probability distribution called your prior distribution
- Suppose you obtain some data relevant to that thing
- The data changes your uncertainty, which is then described by a new probability distribution called your posterior distribution
- Posterior distribution reflects the information both in the prior distribution and the data
- Most of Bayesian inference is about how to go from prior to posterior
- Bayesians go from prior to posterior is to use the laws of conditional probability, sometimes called in this context Bayes rule or Bayes theorem
- Suppose we have a PDF g for the prior distribution of the parameter θ, and suppose we obtain data x whose conditional PDF given θ is f
- Then the joint distribution of data and parameters is conditional times marginal

$$f(x|\theta)g(\theta)$$

- May look strange because most of your training on considers the frequentist paradigm
- Here both x and θ are random variables
- The correct posterior distribution, according to the Bayesian paradigm, is the conditional distribution of θ given x, which is joint divided by marginal

$$h(\theta|x) = \frac{f(x|\theta)g(\theta)}{\int f(x|\theta)g(\theta)d\theta}$$

- Often we do not need to do the integral if we recognize that

  $$\theta \mapsto f(x|\theta)g(\theta)$$

  is, except for constants, the PDF of a brand name distribution, then that distribution must be the posterior

Binomial Data, Beta Prior

Suppose the prior distribution for p is Beta(α_1, α_2) and the conditional distribution of x given p is Bin(n, p). Then

$$f(x|p) = \binom{n}{p} p^x (1-p)^{n-x}$$

and

$$g(p) = \frac{\Gamma(\alpha_1 + \alpha_2)}{\Gamma(\alpha_1)\Gamma(\alpha_2)} p^{\alpha_1 - 1}(1 - p)^{\alpha_2 - 1}.$$

Then

$$f(x|p)g(p) = \binom{n}{p} \frac{\Gamma(\alpha_1 + \alpha_2)}{\Gamma(\alpha_1)\Gamma(\alpha_2)} p^{x + \alpha_1 - 1}(1 - p)^{n - x + \alpha_2 - 1}$$

and this, considered as a function of p for fixed x is, except for constants, the PDF of a Beta$(x+\alpha_1, n-x+\alpha_2)$ distribution. So that is the posterior.

Why?

$$\begin{aligned} h(p|x) &= \frac{f(x|p)g(p)}{\int f(x|p)g(p)dp} \\ &\propto f(x|p)g(p) \\ &= \binom{n}{p} \frac{\Gamma(\alpha_1 + \alpha_2)}{\Gamma(\alpha_1)\Gamma(\alpha_2)} p^{x + \alpha_1 - 1}(1 - p)^{n - x + \alpha_2 - 1} \\ &\propto p^{x + \alpha_1 - 1}(1 - p)^{n - x + \alpha_2 - 1} \end{aligned}$$

And there is only one PDF with support $[0, 1]$ of that form, i.e. a Beta$(x + \alpha_1, n - x + \alpha_2)$ distribution. So that is the posterior.

Bayesian Inference

- In Bayes rule, **constants**, meaning anything that doesn't depend on the parameter, are irrelevant
- We can drop multiplicative constants that do not depend on the parameter from $f(x|\theta)$ obtaining the likelihood $L(\theta)$
- We can also drop multiplicative constants that do not depend on the parameter from $g(\theta)$ obtaining the unnormalized prior
- Multiplying them together gives the unnormalized posterior

$$\text{likelihood } \tilde{\text{A}}^{-} \text{ unnormalized prior } = \text{ unnormalized posterior}$$

In our example we could have multiplied likelihood

$$p^x(1 - p)^{n - x}$$

times unnormalized prior

$$p^{\alpha_1 - 1}(1 - p)^{\alpha_2 - 1}$$

to get unnormalized posterior

$$p^{x + \alpha_1 - 1}(1 - p)^{n - x + \alpha_2 - 1}$$

which, as before, can be recognized as an unnormalized beta PDF.

- It is convenient to have a name for the parameters of the prior and posterior. If we call them parameters, then we get confused because they play a different role from the parameters of the distribution of the data.
- The parameters of the distribution of the data, p in our example, the Bayesian treats as random variables. They are the random variables whose distributions are the prior and posterior.
- The parameters of the prior, α_1 and α_2 in our example, the Bayesian treats as known constants. They determine the particular prior distribution used for a particular problem. To avoid confusion we call them **hyperparameters**.
- Parameters, meaning the parameters of the distribution of the data and the variables of the prior and posterior, are unknown constants. The Bayesian treats them as random variables because probability theory is the correct description of uncertainty.

- Hyperparameters, meaning the parameters of the prior and posterior, are known constants. The Bayesian treats them as non- random variables because there is no uncertainty about their values.
- In our example, the hyperparameters of the prior are α_1 and α_2, and the hyperparameters of the posterior are $x + \alpha_1$ and $n - x + \alpha_2$.

Example: Normal

Suppose $X_1, \ldots, X_n$ are i.i.d. $N(\theta, \sigma^2)$ where σ^2 is known. Suppose further we have a prior $\theta \sim N(\mu, \tau^2)$. Then the posterior can be obtained as follows,

$$\begin{aligned} f(\theta|x) &\propto f(\theta) \prod_{i=1}^{n} f(x_i|\theta) \\ &\propto \exp\left\{ -\frac{1}{2} \left(\frac{(\theta-\mu)^2}{\tau^2} + \frac{\sum_{i=1}^{n}(x_i-\theta)^2}{\sigma^2} \right) \right\} \\ &\propto \exp\left\{ -\frac{1}{2} \frac{\left(\theta - \frac{\mu/\tau^2 + n\bar{x}/\sigma^2}{1/\tau^2 + n/\sigma^2} \right)^2}{\frac{1}{1/\tau^2 + n/\sigma^2}} \right\}. \end{aligned}$$

Or $f(\theta|x) \sim N(\mu_n, \tau_n^2)$ where

$$\mu_n = \left(\frac{\mu}{\tau^2} + \frac{n\bar{x}}{\sigma^2} \right) \tau_n^2 \quad \text{and} \quad \tau_n^2 = \frac{1}{1/\tau^2 + n/\sigma^2}.$$

We will call this a **conjugate** Bayes model. Also note a 95% credible region for θ is given by (this is also the HPD, highest posterior density)

$$(\mu_n - 1.96\tau_n, \mu_n + 1.96\tau_n).$$

For large n, the data will overwhelm the prior.

- If $f(\theta) \propto 1$, an improper prior, then a 95% credible region for θ is the same as a 95% confidence interval since $f(\theta|x) \sim N(\bar{x}, \sigma^2/n)$ (try to show this at home).
- Usually, we specify a prior and likelihood that result in an posterior that is intractable. That is, we can't work with it analytically or even calculate the appropriate normalizing constant c.

- However, it is often easy to simulate a Markov chain with $f(\theta|x)$ as its stationary distribution.

Conjugate Priors

- Given a data distribution $f(\theta|x)$, a family of distributions is said to be conjugate to the given distribution if whenever the prior is in the conjugate family, so is the posterior, regardless of the observed value of the data
- Our first example showed that, if the data distribution is binomial, then the conjugate family of distributions is beta
- Our second example showed that, if the data distribution is normal with known variance, then the conjugate family of distributions is normal

Improper Priors

- A subjective Bayesian is a person who really buys the Bayesian philosophy. Probability is the only correct measure of uncertainty, and this means that people have probability distributions in their heads

that describe any quantities they are uncertain about. In any situation one must make one's best effort to get the correct prior distribution out of the head of the relevant user and into Bayes rule.
- Many people, however, are happy to use the Bayesian paradigm while being much less fussy about priors. When the sample size is large, the likelihood outweighs the prior in determining the posterior. So, when the sample size is large, the prior is not crucial.
- Such people are willing to use priors chosen for mathematical convenience rather than their accurate representation of uncertainty.
- They often use priors that are very spread out to represent extreme uncertainty. Such priors are called "vague" or "diffuse" even though these terms have no precise mathematical definition.
- In the limit as the priors are spread out more and more one gets so-called improper priors.
- There is no guarantee that

$$\text{likelihood } \tilde{\text{A}}^{-} \text{ improper prior } = \text{ unnormalized posterior}$$

results in anything that can be normalized. If the right-hand side integrates, then we get a proper posterior after normalization. If the right-hand does not integrate, then we get complete nonsense.
- You have to be careful when using improper priors that the answer makes sense. Probability theory doesn't guarantee that, because improper priors are not probability distributions.
- Improper priors are questionable
  - Subjective Bayesians think they are nonsense. They do not correctly describe the uncertainty of anyone.
  - Everyone has to be careful using them, because they don't always yield proper posteriors. Everyone agrees improper posteriors are nonsense.
  - Because the joint distribution of data and parameters is also improper, paradoxes arise. These can be puzzling.
- However they are widely used and need to be understood.

Objective Bayesian Inference

- The subjective, personalistic aspect of Bayesian inference bothers many people. Hence many attempts have been made to formulate **objective** priors, which are supposed to be priors that many people can agree on, at least in certain situations.
- However, none of the proposed **objective** priors achieve wide agreement.

Flat Priors

- One obvious **default** prior is flat (constant), which seems to give no preference to any parameter value over any other
- If the parameter space is unbounded, then the flat prior is improper
- One problem with flat priors is that they are only flat for one parameterization
- Another alternative is **Jeffreys priors**

Bayesian Point Estimates

- Bayesians have little interest in point estimates of parameters. To them a parameter is a random variable, and what is important is its distribution. A point estimate is a meager bit of information as compared, for example, to a plot of the posterior density.
- However, Bayesian point estimates are widely reported and something we will be estimating using MCMC
- Bayesian point estimates most commonly used are the posterior mean, the posterior median, the posterior mode, and the endpoints of Bayesian credible regions
- Frequentists too have little interest in point estimates except as tools for constructing tests and confidence intervals

Bayesian Credible Intervals

- Not surprisingly, when a Bayesian makes an interval estimate, it is based on the posterior.
- Many Bayesians do not like to call such things **confidence intervals** because that names a frequentist notion. Hence the name **credible intervals** which is clearly something else.
- One way to make credible intervals is to find the marginal posterior distribution for the parameter of interest and find its $\alpha/2$ and $1 - \alpha/2$ quantiles. The interval between them is a $100(1 - \alpha)\%$ Bayesian credible interval for the parameter of interest called the equal tailed interval.

Bayesian Point Estimates

- Suppose the data x is Bin(n, p) and we use the conjugate prior Beta(α_1, α_2), so the posterior is Beta($x + \alpha_1, n - x + \alpha_2$)
- Since we know the mean of a beta distribution, we can see the posterior mean is

$$E(p|x) = \frac{x + \alpha_1}{x + \alpha_1 + \alpha_2}$$

- The posterior median has no simple expression, but we can calculate it using the R

```
qbeta(0.5, x + alpha1, n - x + alpha2)
```

- The endpoints of Bayesian credible regions can also be found using R, say for an 80% credible region

```
qbeta(0.1, x + alpha1, n - x + alpha2)
qbeta(0.9, x + alpha1, n - x + alpha2)
```

- Suppose $\alpha_1 = \alpha_2 = 1/2$, $x = 2$, and $n = 10$.

```
alpha1 <- alpha2 <- 1/2
x <- 2
n <- 10
(x + alpha1)/(n + alpha1 + alpha2)
```

```
## [1] 0.2272727
```

```
qbeta(0.5, x + alpha1, n - x + alpha1)
```

```
## [1] 0.2103736
```

```
cbind(qbeta(0.1, x + alpha1, n - x + alpha2), qbeta(0.9, x + alpha1, n - x +
    alpha2))
```

```
##            [,1]      [,2]
## [1,] 0.08361516 0.3948296
```

Bayesian Hypothesis Tests

- Not surprisingly, when a Bayesian does a hypothesis test, it is based on the posterior.
- To a Bayesian, a hypothesis is an event, a subset of the sample space. Remember that after the data are seen, the Bayesian considers only the parameter random. So the parameter space and the sample space are the same thing to the Bayesian.
- The Bayesian compares hypotheses by comparing their posterior probabilities.
- All but the simplest such tests must be done by computer.

Bayesian Hypothesis Tests

- Suppose the data x is Bin(n, p) and we use the conjugate prior Beta(α_1, α_2), so the posterior is Beta($x + \alpha_1, n - x + \alpha_2$)
- Suppose the hypotheses in question are

$$H_0 : p \geq 1/2$$
$$H_1 : p < 1/2$$

- We can calculate the probabilities of these two hypotheses by the the R expressions

```
pbeta(0.5, x + alpha1, n - x + alpha2)
```

```
## [1] 0.9739634
```

```
pbeta(0.5, x + alpha1, n - x + alpha2, lower.tail = FALSE)
```

```
## [1] 0.02603661
```

Bayesian Hypothesis Tests

- Suppose $\alpha_1 = \alpha_2 = 1/2$, $x = 2$, and $n = 10$.

```
alpha1 <- alpha2 <- 1/2
x <- 2
n <- 10
pbeta(0.5, x + alpha1, n - x + alpha2)
```

```
## [1] 0.9739634
```

```
pbeta(0.5, x + alpha1, n - x + alpha2, lower.tail = FALSE)
```

```
## [1] 0.02603661
```

Bayesian Hypothesis Tests

- Bayes tests get weirder when the hypotheses have different dimensions
- In principle, there is no reason why a prior distribution has to be continuous
  - It can have degenerate parts that put probability on sets a continuous distribution would give probability zero
  - But many users find this weird
- Bayes Factors tend to be more widely used

Bayes Factors

- Let M be a finite or countable set of models. For each model $m \in M$ we have the prior probability of the model $h(m)$. It does not matter if this prior on models is unnormalized.
- Each model m has a parameter space Θ_m and a prior $g(\theta|m)$, $\theta \in \Theta_m$
- The spaces Θ_m can and usually do have different dimensions. That's the point. These within model priors must be normalized proper priors. The calculations to follow make no sense if these priors are unnormalized or improper.
- Each model m has a data distribution

$$f(x|\theta, m)$$

  which may be a PDF or PMF.

The unnormalized posterior for everything, models and parameters within models, is

$$f(x|\theta, m)g(\theta|m)h(m)$$

To obtain the conditional distribution of x given m, we must integrate out the nuisance parameters θ

$$q(x|m) = \int_{\Theta_m} f(x|\theta, m)g(\theta|m)h(m)d\theta$$
$$h(m) \int_{\Theta_m} f(x|\theta, m)g(\theta|m)d\theta$$

These are the unnormalized posterior probabilities of the models.

The normalized probabilities are

$$p(m|x) = \frac{q(x|m)}{\sum q(x|m)}$$

It is useful to define

$$b(x|m) = \int_{\Theta_m} f(x|\theta, m)g(\theta|m)d\theta$$

so

$$q(x|m) = b(x|m)h(m)$$

Then the ratio of posterior probabilities of models m_1 and m_2 is

$$\frac{p(m_1|x)}{p(m_2|x)} = \frac{q(x|m_1)}{q(x|m_2)} = \frac{b(x|m_1)h(m_1)}{b(x|m_2)h(m_2)}$$

This ratio is called the **posterior odds** of the models (a ratio of probabilities is called an odds) of these models.

The **prior odds** is

$$\frac{h(m_1)}{h(m_2)}$$

The term we have not yet named in

$$\frac{p(m_1|x)}{p(m_2|x)} = \frac{b(x|m_1)h(m_1)}{b(x|m_2)h(m_2)}$$

is called the **Bayes factor**

$$\frac{b(x|m_1)}{b(x|m_2)}$$

the ratio of posterior odds to prior odds.

The prior odds tells how the prior compares the probability of the models. The Bayes factor tells us how the data shifts that comparison going from prior to posterior via Bayes rule.

- Suppose the data x is Bin(n, p) and the models (hypotheses) in question are

$$m_1 : p = 1/2$$
$$m_2 : p \neq 1/2$$

- The model m_1 is concentrated at one point $p = 1/2$, hence has no nuisance parameter. Hence $g(\theta|m_1) = 1$. Suppose we use the within model prior Beta(α_1, α_2) for model m_2.
- Then

$$b(x|m_1) = f(x|1/2) = \binom{n}{x}(1/2)^x(1 - 1/2)^{n-x} = \binom{n}{x}(1/2)^n$$

Then

$$\begin{aligned} b(x|m_2) &= \int_0^1 f(x|p)g(p|m_2)dp \\ &= \int_0^1 \binom{n}{x} \frac{1}{B(\alpha_1, \alpha_2)} p^{x+\alpha_1-1}(1-p)^{n-x+\alpha_2-1} dp \\ &= \binom{n}{x} \frac{B(x+\alpha_1, n-x+\alpha_2)}{B(\alpha_1, \alpha_2)} \end{aligned}$$

where

$$B(\alpha_1, \alpha_2) = \frac{\Gamma(\alpha_1)\Gamma(\alpha_2)}{\Gamma(\alpha_1+\alpha_2)}$$

by properties of the Beta distribution

```
alpha1 <- alpha2 <- 1/2
x <- 2
n <- 10
p0 <- 1/2
b1 <- dbinom(x, n, p0)
b2 <- choose(n, x) * beta(x + alpha1, n - x + alpha2)/beta(alpha1, alpha2)
BayesFactor <- b1/b2
BayesFactor
```

```
## [1] 0.5967366
```

```
pvalue <- 2 * pbinom(x, n, p0)
pvalue
```

```
## [1] 0.109375
```

- For comparison, we calculated not only the Bayes factor 0.597 but also the frequentist p-value 0.109
  - Bayes factors and p-values are sort of comparable, but are **not** identical
- In fact, it is a theorem that in situations like this the Bayes factor is always larger than the p-value, at least asymptotically
- This makes Bayesian tests more conservative, less likely to reject the null hypothesis, than frequentists
- Either the frequentists are too optimistic or the Bayesians are too conservative, or perhaps both

Summary

- Bayesian inference, priors, Bayesian point estimates, Bayesian hypothesis testing, and Bayes factors
- Many applications including pattern recognition, span detection, search for lost objects, ...
- Calculations are trivial in our examples so far, not usually the case

Lecture 16: Markov Chain Monte Carlo I

Agenda

- Like Ordinary Monte Carlo (OMC), but better?
- SLLN and Markov chain CLT
- Variance estimation
- AR(1) example
- Metropolis-Hastings algorithm (with an exercise)

Markov chain Monte Carlo

- A Markov chain is a dependent sequence of random variables $X_1, X_2, \ldots$ or or random vectors $X_1, X_2, \ldots$ having the property that the future is independent of the past given the present
- Conditional distribution of X_{n+1} given $X_1, \ldots, X_n$ depends only on X_n
- The Markov chain has **stationary transition probabilities** if the conditional distribution of X_{n+1} given X_n is the same for all n
  - Every Markov chain used in MCMC has this property
- The joint distribution of $X_1, \ldots, X_n$ is determined by the initial distribution of the Markov chain and the transition probabilities
  - Marginal distribution of X_1
  - Conditional distribution of X_{n+1} given X_n
- A scalar functional of a Markov chain is a time series, but not necessarily a Markov chain
- A Markov chain is **stationary** if its initial distribution is stationary
  - Different from having stationary transition probabilities
  - All chains used in MCMC have stationary transition probabilities, but none are exactly stationary
- To be (exactly) stationary, must start the chain with simulation from the equilibrium (invariant, stationary) distribution
- If chain is stationary, then every iterate X_i has the same marginal distribution, which is the equilibrium distribution
- If chain is not stationary but has a unique equilibrium distribution, which includes chains used in MCMC, then the marginal distribution X_i converges to the equilibrium distribution as $i \to \infty$
- Let π be a probability distribution having support $\mathcal{X} \subseteq \mathbb{R}^d$, $d \geq 1$ we want to explore
- When i.i.d. observations are unavailable, a Markov chain with stationary distribution π can be utilized
- Summarize π with expectations, quantiles, density plots ...

- Suppose $X_1, \ldots, X_n$ are simulation from a Markov chain having a unique equilibrium distribution (say π), and suppose we want to know an expectation

$$\mu_g = E[g(X_i)] = \int_{\mathcal{X}} g(x)\, \pi(dx)$$

  where the expectation is with respect to unique equilibrium distribution π
- If $E_\pi |g(X_i)| < \infty$, then

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^{n} g(X_i) \overset{a.s.}{\to} \mu_g \quad \text{as } n \to \infty \; (SLLN).$$

- The central limit theorem (CLT) for Markov chains says

$$\sqrt{n}(\hat{\mu}_n - E_\pi g(X_i)) \to \mathrm{N}(0, \sigma^2)\ ,$$

  where

$$\sigma^2 = \mathrm{Var} g(X_i) + 2 \sum_{k=1}^{\infty} \mathrm{Cov}\left[g(X_i), g(X_{i+k})\right]$$

- CLT holds if $E_\pi|g(X_i)|^{2+\epsilon} < \infty$ and the Markov chain is geometrically ergodic
- Can estimate σ^2 in various ways
- Verifying such a mixing condition is generally very challenging
- Nevertheless, we expect the CLT to hold in practice when using a **smart** sampler

Batch means

- In order to make MCMC practical, need a method to estimate the variance σ^2 in the CLT, then can proceed just like in OMC
- If $\hat{\sigma}^2$ is a consistent estimate of σ^2, then an asymptotic 95% confidence interval for μ_g is

$$\hat{\mu}_n \pm 1.96\frac{\hat{\sigma}}{\sqrt{n}}$$

- The method of batch means estimates the asymptotic variance for a stationary time series
- Markov chain CLT says

$$\hat{\mu}_n \approx \mathrm{N}\left(\mu_g, \frac{\sigma^2}{n}\right)$$

- Suppose b evenly divides n and we have the means

$$\hat{\mu}_{b,k} = \frac{1}{b}\sum_{i=bk+1}^{bk+b} g(X_i)$$

  for $k = 1, \ldots, a = n/b$
- Then each of these **batch means** satisfies (if b is sufficiently large)

$$\hat{\mu}_{b,k} \approx \mathrm{N}\left(\mu_g, \frac{\sigma^2}{b}\right)$$

- Thus empirical variance of the sequence of batch means

$$\frac{1}{a}\sum_{k=1}^{a}\left(\hat{\mu}_{b,k} - \hat{\mu}_n\right)^2$$

  estimates σ^2/b
- And b/n times this estimates σ^2/n, the asymptotic variance of $\hat{\mu}_n$
- Batch means can produce a strongly consistent estimator of σ^2 if $b \to \infty$ and $a \to \infty$ as $n \to \infty$

Stopping rules

- Suppose $\epsilon > 0$, then a **fixed-width stopping rule* terminates the simulation the first time half-width (or width) of a confidence interval is sufficiently small
- That is, simulate until

$$1.96\frac{\hat{\sigma}}{\sqrt{n}} < \epsilon.$$

Example: AR(1)

- Consider the Markov chain such that

$$X_i = \rho X_{i-1} + \epsilon_i$$

  where $\epsilon_i \overset{iid}{\sim} N(0,1)$

- Consider $X_1 = 0$, $\rho = .95$, and estimating $E_\pi X = 0$
- Run until

$$w_n = 2z_{.975}\frac{\hat{\sigma}}{\sqrt{n}} \leq 0.2$$

where $\hat{\sigma}$ is calculated using batch means

The following will provide an observation from the MC 1 step ahead

```
ar1 <- function(m, rho, tau) {
    rho * m + rnorm(1, 0, tau)
}
```

Next, we add to this function so that we can give it a Markov chain and the result will be p observations from the Markov chain

```
ar1.gen <- function(mc, p, rho, tau, q = 1) {
    loc <- length(mc)
    junk <- double(p)
    mc <- append(mc, junk)

    for (i in 1:p) {
        j <- i + loc - 1
        mc[(j + 1)] <- ar1(mc[j], rho, tau)
    }
    return(mc)
}
```

```
set.seed(20)
library(mcmcse)
```

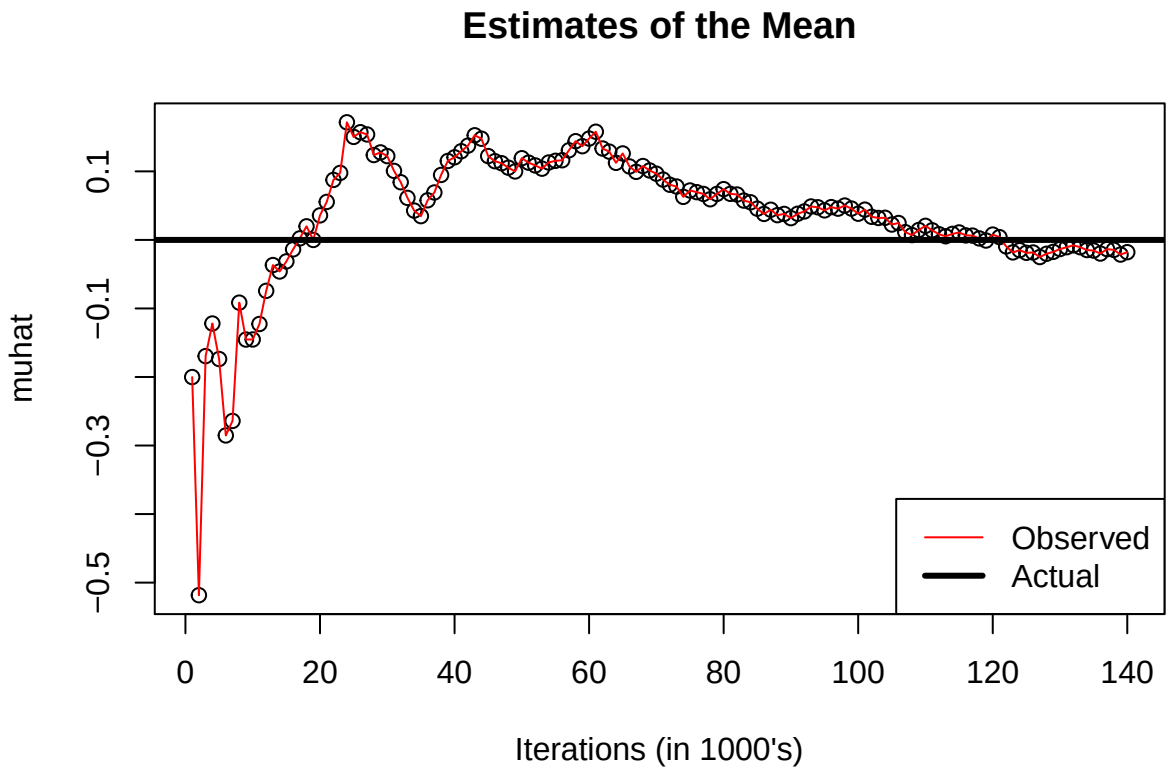
```
## mcmcse: Monte Carlo Standard Errors for MCMC
## Version 1.2-1 created on 2016-03-24.
## copyright (c) 2012, James M. Flegal, University of California,Riverside
##                     John Hughes, University of Minnesota
##                     Dootika Vats, University of Minnesota
##  For citation information, type citation("mcmcse").
##  Type help("mcmcse-package") to get started.
```

```
tau <- 1
rho <- 0.95
out <- 0
eps <- 0.1
start <- 1000
r <- 1000
```

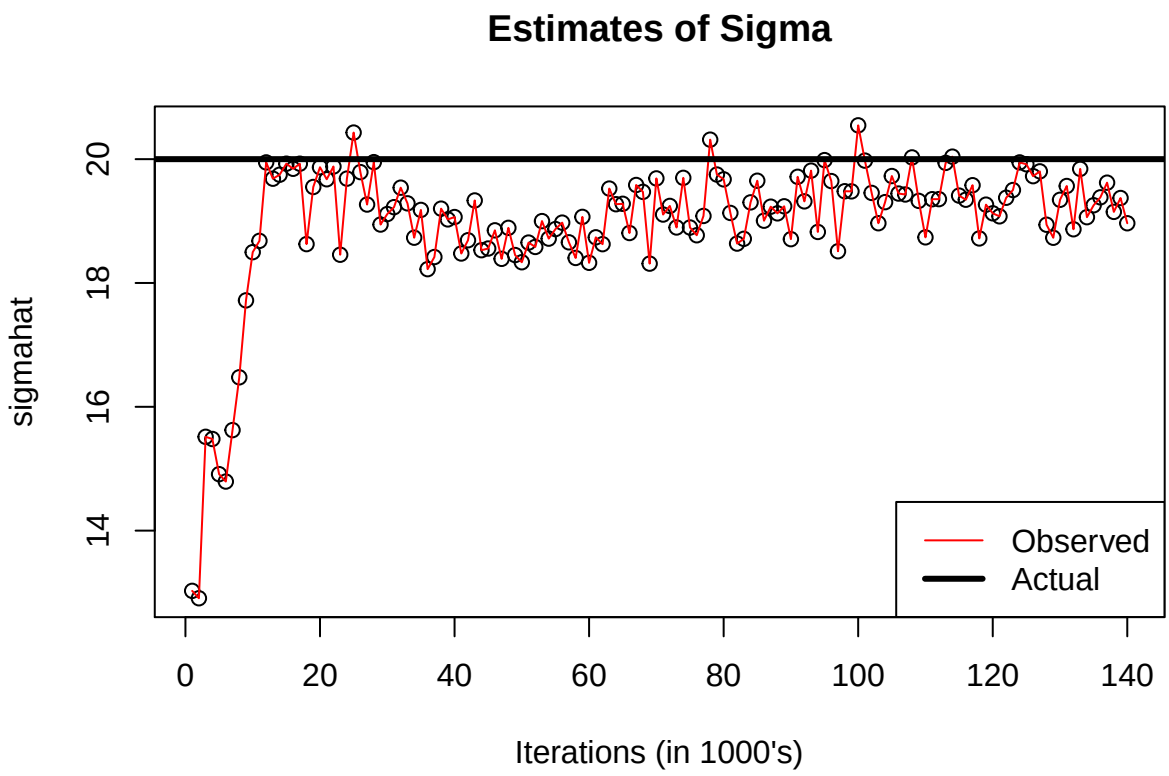
```
out <- ar1.gen(out, start, rho, tau)
MCSE <- mcse(out)$se
N <- length(out)
t <- qt(0.975, (floor(sqrt(N) - 1)))
muhat <- mean(out)
check <- MCSE * t

while (eps < check) {
    out <- ar1.gen(out, r, rho, tau)
    MCSE <- append(MCSE, mcse(out)$se)
    N <- length(out)
    t <- qt(0.975, (floor(sqrt(N) - 1)))
```

```
    muhat <- append(muhat, mean(out))
    check <- MCSE[length(MCSE)] * t
}

N <- seq(start, length(out), r)
t <- qt(0.975, (floor(sqrt(N) - 1)))
half <- MCSE * t
sigmahat <- MCSE * sqrt(N)
N <- seq(start, length(out), r)/1000
```

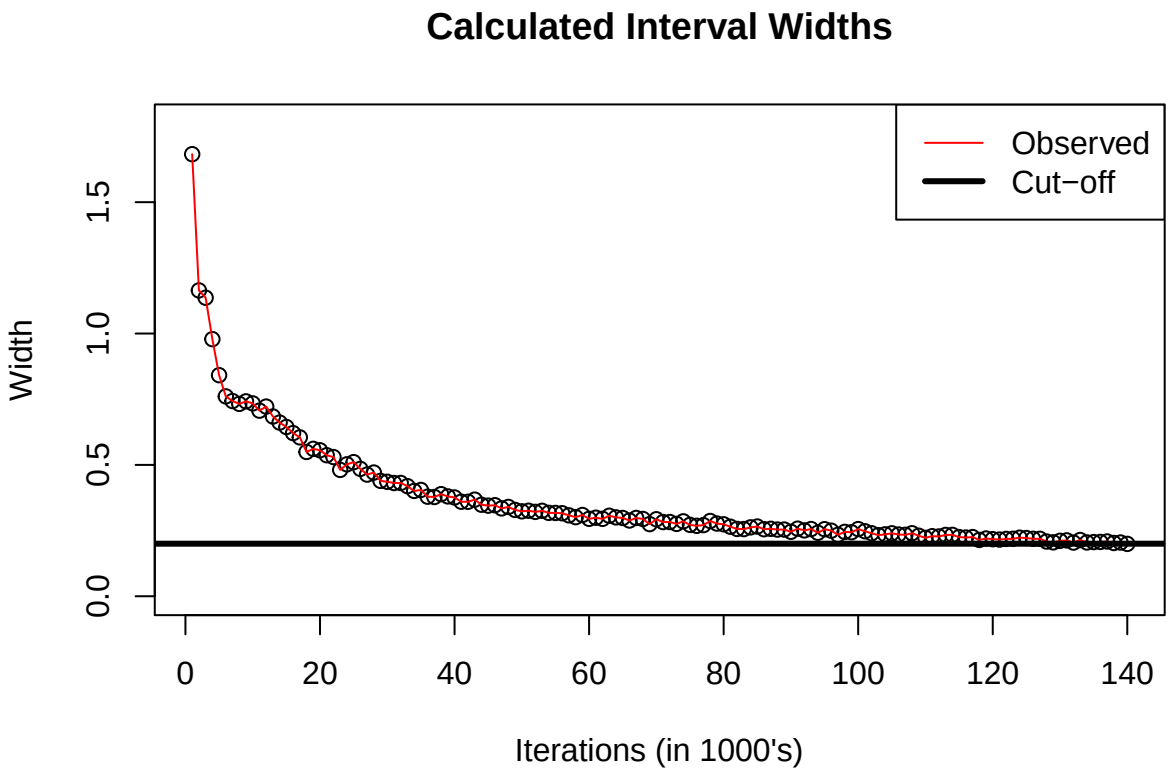
```
plot(N, muhat, main = "Estimates of the Mean", xlab = "Iterations (in 1000's)")
points(N, muhat, type = "l", col = "red")
abline(h = 0, lwd = 3)
legend("bottomright", legend = c("Observed", "Actual"), lty = c(1, 1), col = c(2,
    1), lwd = c(1, 3))
```




```
plot(N, sigmahat, main = "Estimates of Sigma", xlab = "Iterations (in 1000's)")
points(N, sigmahat, type = "l", col = "red")
abline(h = 20, lwd = 3)
legend("bottomright", legend = c("Observed", "Actual"), lty = c(1, 1), col = c(2,
    1), lwd = c(1, 3))
```




```
plot(N, 2 * half, main = "Calculated Interval Widths", xlab = "Iterations (in 1000's)",
    ylab = "Width", ylim = c(0, 1.8))
points(N, 2 * half, type = "l", col = "red")
abline(h = 0.2, lwd = 3)
legend("topright", legend = c("Observed", "Cut-off"), lty = c(1, 1), col = c(2,
    1), lwd = c(1, 3))
```



Markov chain Monte Carlo

- MCMC methods are used most often in Bayesian inference where the equilibrium (invariant, stationary) distribution is a posterior distribution
- Challenge lies in construction of a suitable Markov chain with f as its stationary distribution
- A key problem is we only get to observe t observations from $\{X_t\}$, which are serially **dependent**
- Other questions to consider
  - How good are my MCMC estimators?
  - How long to run my Markov chain simulation?
  - How to compare MCMC samplers?
  - What to do in high-dimensional settings?
  - ...

Metropolis-Hastings algorithm

Setting $X_0 = x_0$ (somehow), the Metropolis-Hastings algorithm generates X_{t+1} given $X_t = x_t$ as follows:

1. Sample a candidate value $X^* \sim g(\cdot|x_t)$ where g is the proposal distribution
2. Compute the MH ratio $R(x_t, X^*)$, where

$$R(x_t, X^*) = \frac{f(x^*)g(x_t|x^*)}{f(x_t)g(x^*|x_t)}$$

3. Set

$$X_{t+1} = \begin{cases} x^* \text{ w.p. } \min\{R(x_t, X^*), 1\} \\ x_t \text{ otherwise} \end{cases}$$

- Irreducibility and aperiodicity depend on the choice of g, these must be checked
- Performance (finite sample) depends on the choice of g also, be careful

Independence chains

- Suppose $g(x^*|x_t) = g(x^*)$, this yields an **independence** chain since the proposal does not depend on the current state
- In this case, the MH ratio is

$$R(x_t, X^*) = \frac{f(x^*)g(x_t)}{f(x_t)g(x^*)},$$

  and the resulting Markov chain will be irreducible and aperiodic if $g > 0$ where $f > 0$
- A good envelope function g should resemble f, but should cover f in the tails

Random walk chains

- Generate X^* such that $\epsilon \sim h(\cdot)$ and set $X^* = X_t + \epsilon$, then $g(x^*|x_t) = h(x^* - x_t)$
- Common choices of $h(\cdot)$ are symmetric zero mean random variables with a scale parameter, e.g. a Uniform$(-a, a)$, Normal$(0, \sigma^2)$, $c * T_\nu, \ldots$
- For symmetric zero mean random variables, the MH ratio is

$$R(x_t, X^*) = \frac{f(x^*)}{f(x_t)}$$

- If the support of f is connected and h is positive in a neighborhood of 0, then the chain is irreducible and aperiodic.

Example: Markov chain basics

Exercise: Suppose $f \sim Exp(1)$

1. Write an independence MH sampler with $g \sim Exp(\theta)$
2. Show $R(x_t, X^*) = \exp\{(x_t - x^*)(1 - \theta)\}$
3. Generate 1000 draws from f with $\theta \in \{1/2, 1, 2\}$
4. Write a random walk MH sampler with $h \sim N(0, \sigma^2)$
5. Show $R(x_t, X^*) = \exp\{x_t - x^*\} I(x^* > 0)$
6. Generate 1000 draws from f with $\sigma \in \{.2, 1, 5\}$
7. In general, do you prefer an independence chain or a random walk MH sampler? Why?
8. Implement the fixed-width stopping rule for you preferred chain

Independence Metropolis sampler with Exp(θ) proposal

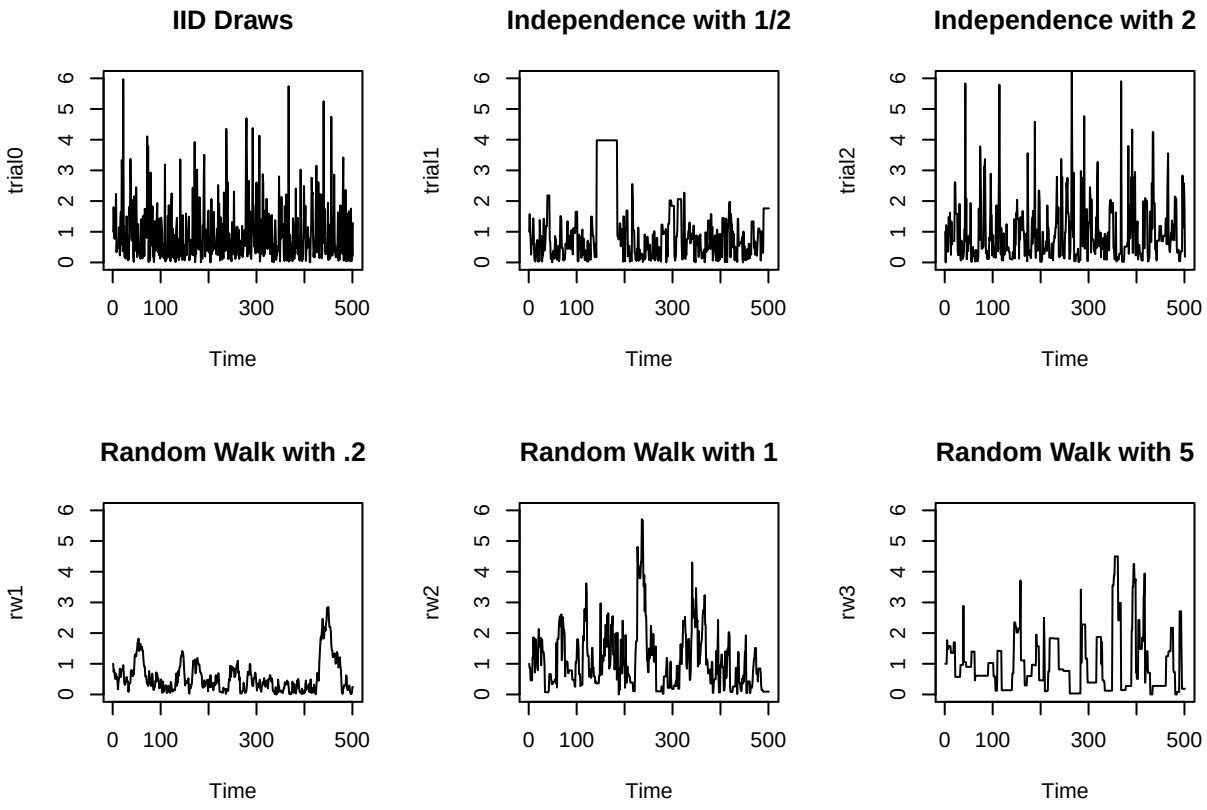
```
ind.chain <- function(x, n, theta = 1) {
    ## if theta = 1, then this is an iid sampler
    m <- length(x)
    x <- append(x, double(n))
    for (i in (m + 1):length(x)) {
        x.prime <- rexp(1, rate = theta)
        u <- exp((x[(i - 1)] - x.prime) * (1 - theta))
        if (runif(1) < u)
            x[i] <- x.prime else x[i] <- x[(i - 1)]
```

```
    }
    return(x)
}
```

Random Walk Metropolis sampler with $\mathrm{N}(0, \sigma)$ proposal

```
rw.chain <- function(x, n, sigma = 1) {
    m <- length(x)
    x <- append(x, double(n))
    for (i in (m + 1):length(x)) {
        x.prime <- x[(i - 1)] + rnorm(1, sd = sigma)
        u <- exp((x[(i - 1)] - x.prime))
        u
        if (runif(1) < u && x.prime > 0)
            x[i] <- x.prime else x[i] <- x[(i - 1)]
    }
    return(x)
}
```

```
trial0 <- ind.chain(1, 500, 1)
trial1 <- ind.chain(1, 500, 2)
trial2 <- ind.chain(1, 500, 1/2)
rw1 <- rw.chain(1, 500, 0.2)
rw2 <- rw.chain(1, 500, 1)
rw3 <- rw.chain(1, 500, 5)
```



Lecture 17: Markov Chain Monte Carlo II

Agenda

- Markov chain Monte Carlo, again
- Gibbs sampling
- Output analysis for MCMC
- Convergence diagnostics
- Examples: Capture-recapture and toy example

Gibbs Sampling

1. Select starting values x_0 and set $t = 0$
2. Generate in turn (deterministic scan Gibbs sampler)
   - $x_{t+1}^{(1)} \sim f(x^{(1)}|x_t^{(-1)})$
   - $x_{t+1}^{(2)} \sim f(x^{(2)}|x_{t+1}^{(1)}, x_t^{(3)}, \ldots, x_t^{(p)})$
   - $x_{t+1}^{(3)} \sim f(x^{(3)}|x_{t+1}^{(1)}, x_{t+1}^{(2)}, x_t^{(4)}, \ldots, x_t^{(p)})$
   - $\ldots$
   - $x_{t+1}^{(p)} \sim f(x^{(p)}|x_{t+1}^{(-p)})$
3. Increment t and go to Step 2

- Common to have one or more components not available in closed form
- Then one can just use a MH sampler for those components known as a Metropolis within Gibbs or Hybrid Gibbs sampling
- Common to `block` groups of random variables

Example: Capture-recapture

- Data from a fur seal pup capture-recapture study for $i = 7$ census attempts
- Goal is to estimate the number of pups in a fur seal colony using a capture-recapture study

Count	Parameter	1	2	3	4	5	6	7
Captured	c_i	30	22	29	26	31	32	35
Newly Caught	m_i	30	8	17	7	9	8	5

- Let N be the population size, I be the number of census attempts where c_i were captured ($I = 7$ in our case), and r be the total number captured ($r = \sum_{i=1}^{I} m_i = 84$)
- We consider a separate unknown capture probability for each census $(\alpha_1, \ldots, \alpha_I)$ where the animals are equally "catchable"
- Then

$$L(N, \alpha|c, r) \propto \frac{N!}{(N-r)!} \prod_{i=1}^{I} \alpha_i^{c_i} (1-\alpha_i)^{N-c_i}$$

- Assume N and α are apriori independent with

$$f(N) \propto 1 \text{ and } f(\alpha_i|\theta_1, \theta_2) \overset{i.i.d.}{\sim} \text{Beta}(\theta_1, \theta_2)$$

- We use $\theta_1 = \theta_2 = 1/2$, which is the Jeffrey's Prior
- The resulting posterior is proper when $I > 2$ and recommended when $I > 5$

- Then it is easy to show the posterior is

$$f(N, \alpha | c, r) \propto \frac{N!}{(N-r)!} \prod_{i=1}^{I} \alpha_i^{c_i} (1-\alpha_i)^{N-c_i} \prod_{i=1}^{I} \alpha_i^{-1/2} (1-\alpha_i)^{-1/2}.$$

- Further, one can show

$$N - 84 | \alpha \sim \text{NB}\left(85, 1 - \prod_{i=1}^{I} (1-\alpha_i)\right) \text{ and}$$
$$\alpha_i | N \sim \text{Beta}\left(c_i + 1/2, N - c_i + 1/2\right) \text{ for all } i$$

Then we can consider the chain

$$(N, \alpha) \to (N', \alpha) \to (N', \alpha')$$

or

$$(N, \alpha) \to (N, \alpha') \to (N', \alpha'),$$

where both involve a "block" update of α

First, we can write the data into R

```
captured <- c(30, 22, 29, 26, 31, 32, 35)
new.captures <- c(30, 8, 17, 7, 9, 8, 5)
total.r <- sum(new.captures)
```

The following R code implements the Gibbs sampler

```
gibbs.chain <- function(n, N.start = 94, alpha.start = rep(0.5, 7)) {
    output <- matrix(0, nrow = n, ncol = 8)
    for (i in 1:n) {
        neg.binom.prob <- 1 - prod(1 - alpha.start)
        N.new <- rnbinom(1, 85, neg.binom.prob) + total.r
        beta1 <- captured + 0.5
        beta2 <- N.new - captured + 0.5
        alpha.new <- rbeta(7, beta1, beta2)
        output[i, ] <- c(N.new, alpha.new)
        N.start <- N.new
        alpha.start <- alpha.new
    }
    return(output)
}
```

MCMC output analysis

How can we tell if the chain is mixing well?

- Trace plots or time-series plots
- Autocorrelation plots
- Plot of estimate versus Markov chain sample size
- Effective sample size (ESS)

$$\text{ESS}(n) = \frac{n}{1 + 2\sum_{k=1}^{\infty} \rho_k(g)},$$

  where $\rho_k(g)$ is the autocorrelation of lag k for g
- Alternative, ESS can be written as

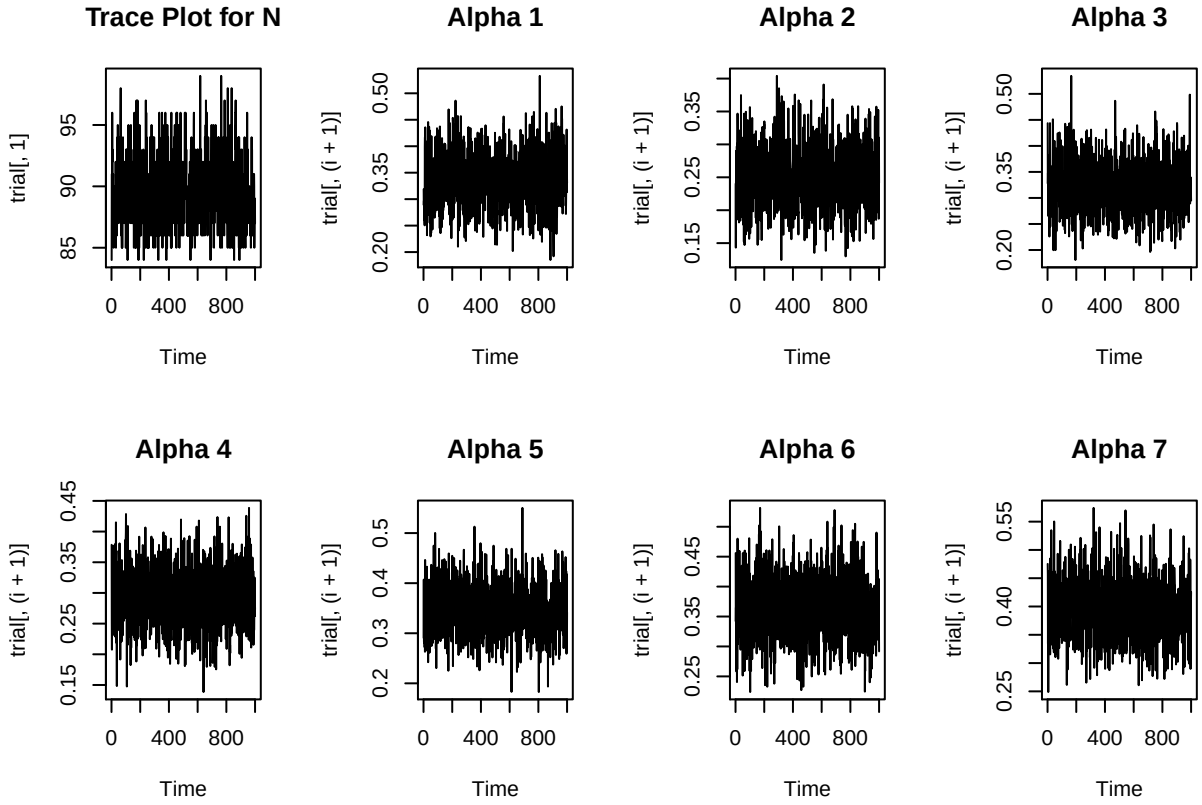
$$\text{ESS}(n) = \frac{n}{\sigma^2 / \text{Var} g}$$

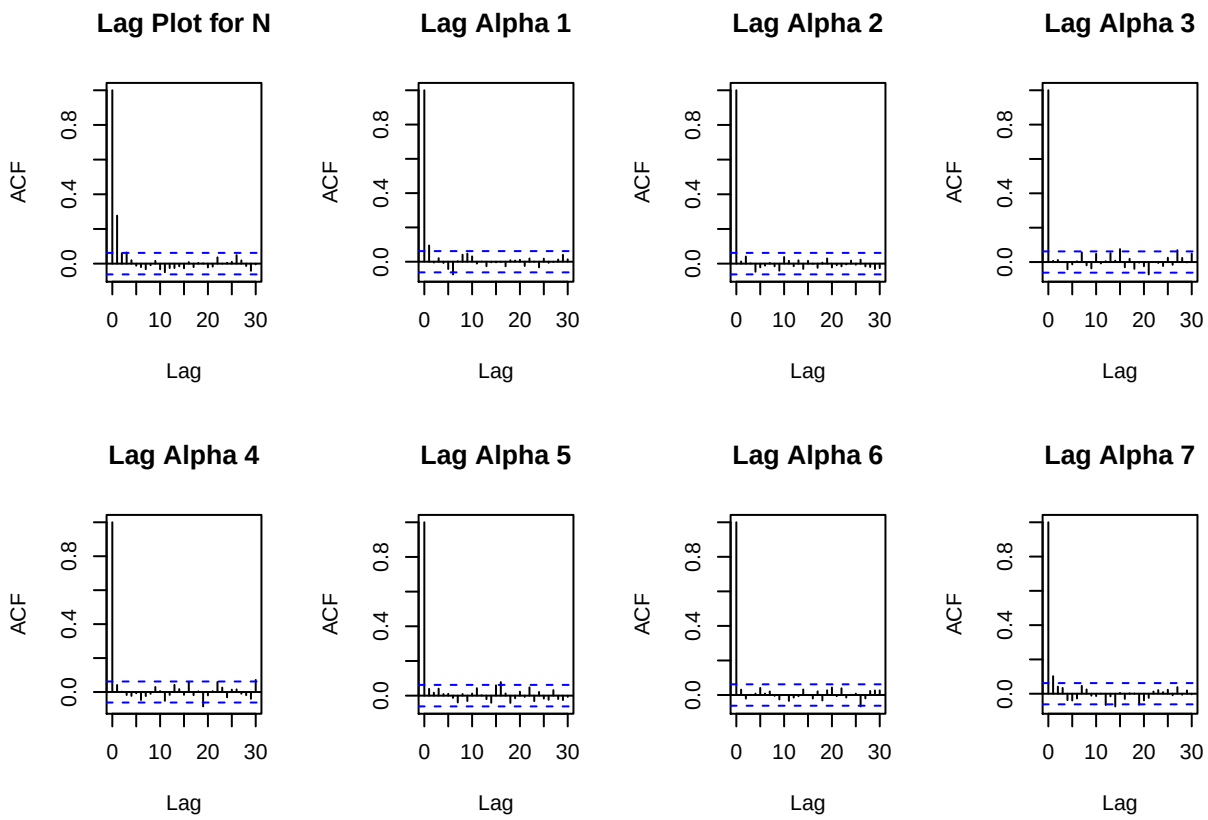
  where σ^2 is the asymptotic variance from a Markov chain CLT

Example: Capture-recapture

Then we consider some preliminary simulations to ensure the chain is mixing well

```
trial <- gibbs.chain(1000)
plot.ts(trial[, 1], main = "Trace Plot for N")
for (i in 1:7) {
    plot.ts(trial[, (i + 1)], main = paste("Alpha", i))
}

acf(trial[, 1], main = "Lag Plot for N")
for (i in 1:7) {
    acf(trial[, (i + 1)], main = paste("Lag Alpha", i))
}
```

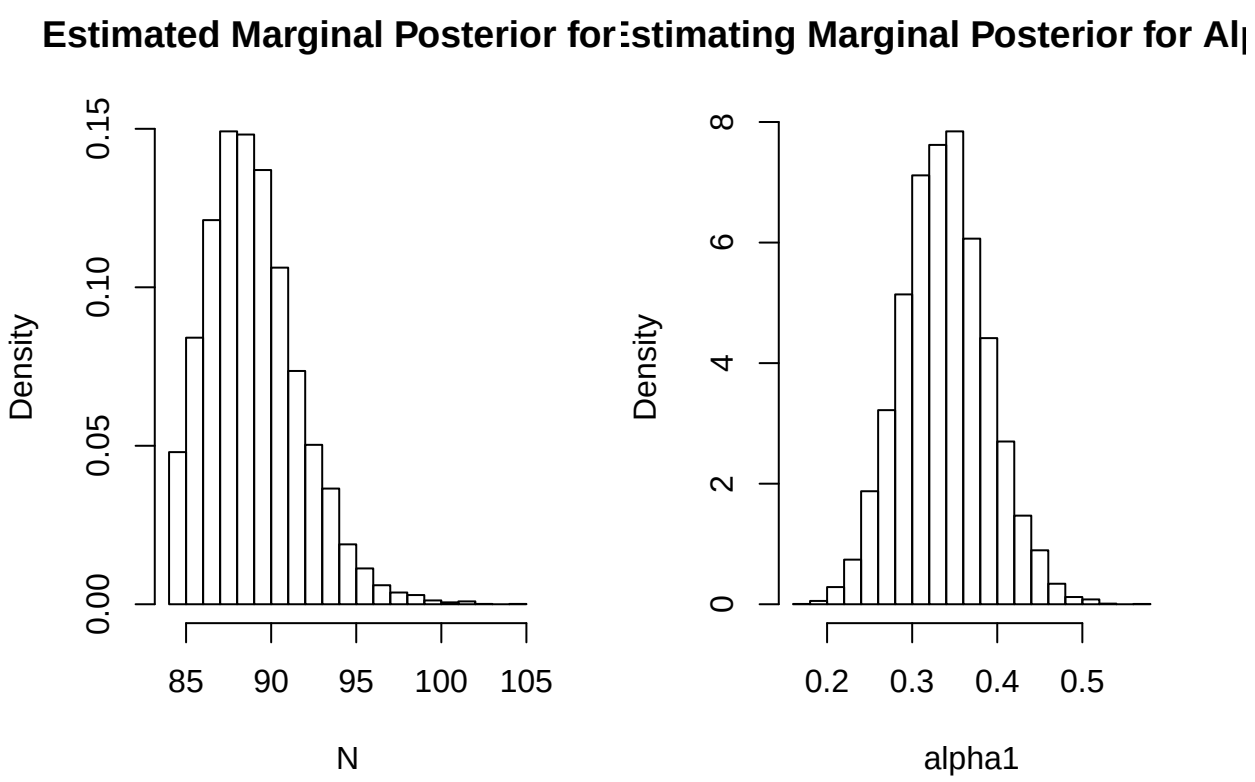




Now for a more complete simulation to estimate posterior means and a 90% Bayesian credible region

```
sim <- gibbs.chain(10000)
N <- sim[, 1]
alpha1 <- sim[, 2]
```

```
par(mfrow = c(1, 2))
hist(N, freq = F, main = "Estimated Marginal Posterior for N")
hist(alpha1, freq = F, main = "Estimating Marginal Posterior for Alpha 1")
```




```
par(mfrow = c(1, 1))
```

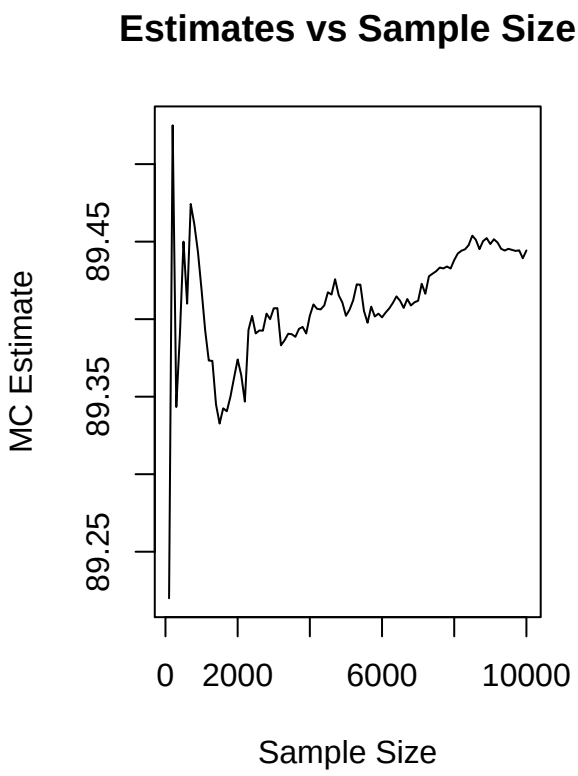
```
library(mcmcse)
ess(N)
```

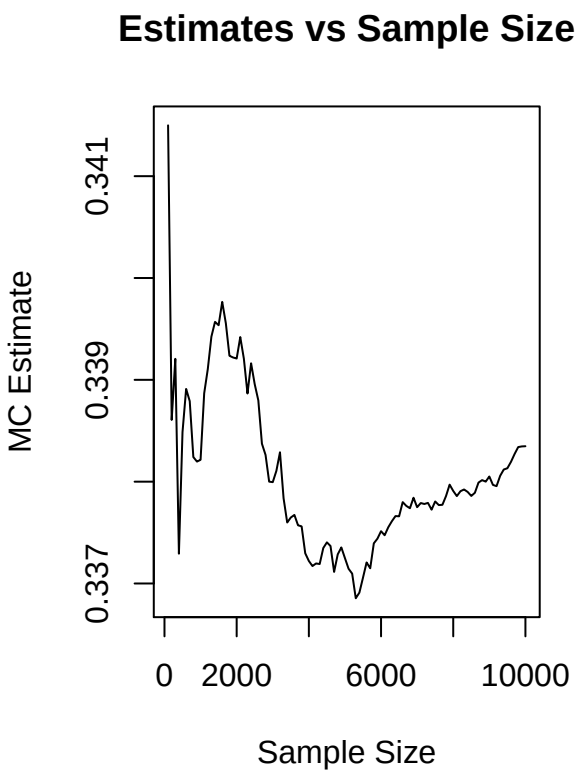
```
##       se
## 7017.894
```

```
ess(alpha1)
```

```
##       se
## 8835.873
```

```
par(mfrow = c(1, 2))
estvssamp(N)
estvssamp(alpha1)
```






```
par(mfrow = c(1, 1))
```

```
mcse(N)
```

```
## $est
## [1] 89.4443
##
## $se
## [1] 0.03300513
```

```
mcse.q(N, 0.05)
```

```
## $est
## [1] 86
##
## $se
## [1] 0.04042096
```

```
mcse.q(N, 0.95)
```

```
## $est
## [1] 94
##
## $se
## [1] 0.05330754
```

```
mcse(alpha1)
```

```
## $est
## [1] 0.3383478
##
## $se
```

```
## [1] 0.0005403667
```

```
mcse.q(alpha1, 0.05)
```

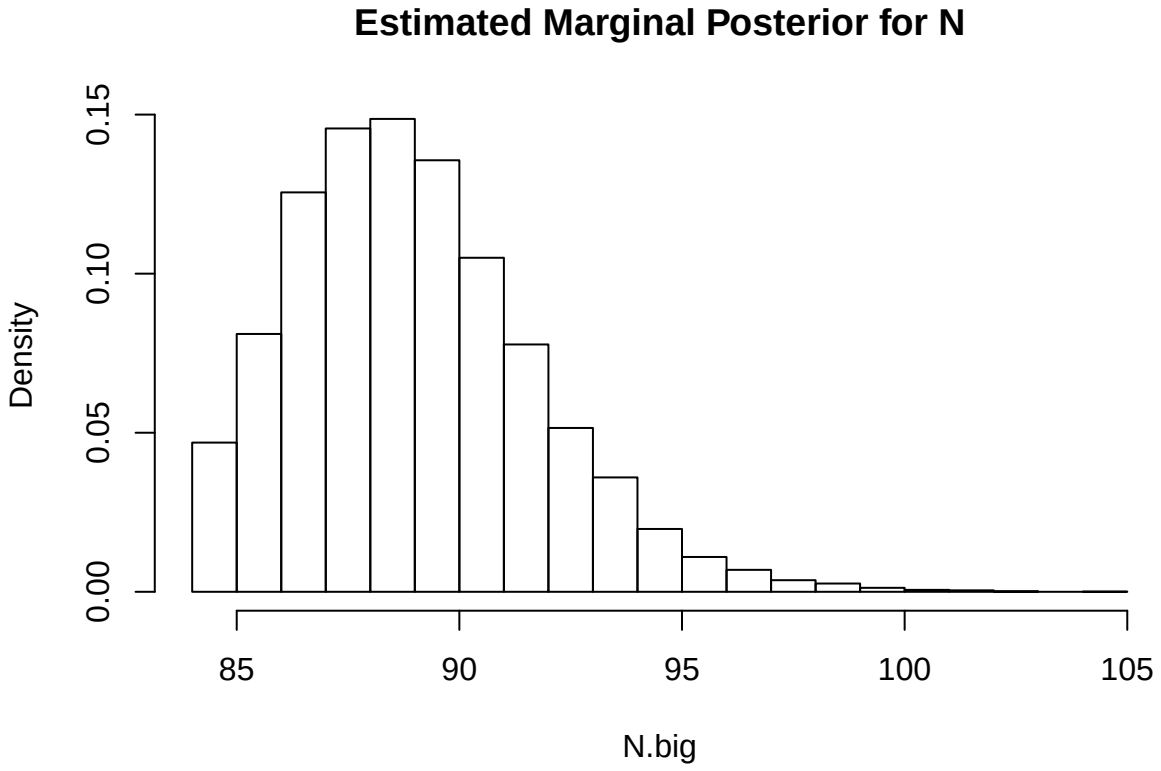
```
## $est
## [1] 0.2559985
##
## $se
## [1] 0.00105318
```

```
mcse.q(alpha1, 0.95)
```

```
## $est
## [1] 0.4241873
##
## $se
## [1] 0.001300649
```

```
current <- sim[10000, ]  # start from here is you need more simulations
sim <- rbind(sim, gibbs.chain(10000, N.start = current[1], alpha.start = current[2:8]))
N.big <- sim[, 1]
```

```
hist(N.big, freq = F, main = "Estimated Marginal Posterior for N")
```



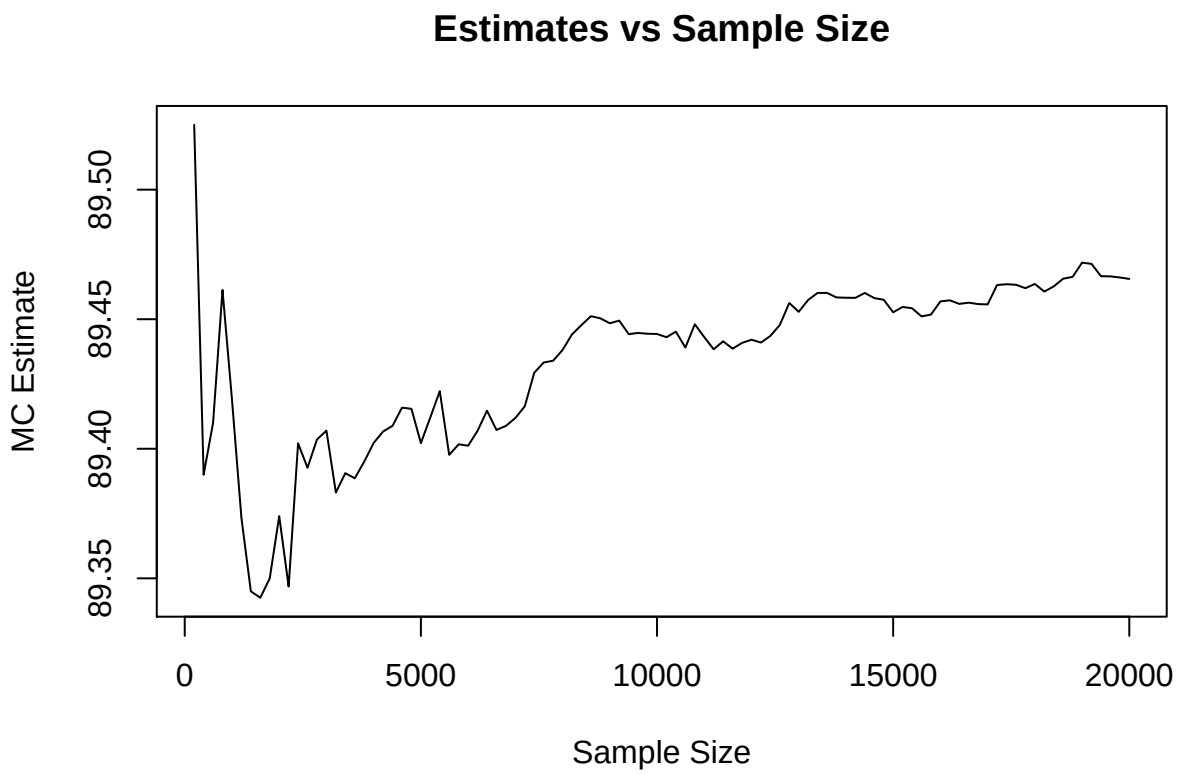

```
ess(N)
```

```
##       se
## 7017.894
```

```
ess(N.big)
```

```
##       se
## 15892.23
```

```
estvssamp(N.big)
```




```
mcse(N)
```

```
## $est
## [1] 89.4443
##
## $se
## [1] 0.03300513
```

```
mcse(N.big)
```

```
## $est
## [1] 89.46555
##
## $se
## [1] 0.02184623
```

```
mcse.q(N, 0.05)
```

```
## $est
## [1] 86
##
## $se
## [1] 0.04042096
```

```
mcse.q(N.big, 0.05)
```

```
## $est
## [1] 86
##
## $se
```

```
## [1] 0.02651976
```

```
mcse.q(N, 0.95)
```

```
## $est
## [1] 94
##
## $se
## [1] 0.05330754
```

```
mcse.q(N.big, 0.95)
```

```
## $est
## [1] 94
##
## $se
## [1] 0.03582395
```

Convergence diagnostics

- A more popular method of MCMC output analysis
- There are many; Gelman and Rubin diagnostic, Geweke's diagnostic, Heidel diagnostic, Raferty diagnostic, and other in `coda` package
- Useful to detect problem with a sampler, but offer ***no*** guarantee you have converged
- Think of these like hypothesis testing

Gelman and Rubin diagnostic

Gelman and Rubin Diagnostic is also used a stopping criteria

- Most popular method for stopping the simulation, one of many convergence diagnostics
- Simulates m independent parallel Markov chains
- Considers a ratio of two different estimates of $\text{Var}_\pi g$, not σ_g^2 from the CLT
- Argue the simulation should continue until the diagnostic ($\widehat{R}_{0.975}$) is close to 1

Toy example

- Let $Y_1, \ldots, Y_m$ be i.i.d. $\text{N}(\mu, \lambda)$ and let the prior for (μ, λ) be proportional to $1/\sqrt{\lambda}$
- The posterior density is characterized by

$$\pi(\mu, \lambda | y) \propto \lambda^{-\frac{m+1}{2}} \exp\left\{ -\frac{1}{2\lambda} \sum_{j=1}^{m} (y_j - \mu)^2 \right\}$$

  which is proper as long as $m \geq 3$

- A Gibbs sampler requires the full conditionals

$$\mu | \lambda, y \sim \text{N}(\bar{y}, \lambda/m)$$

  and

$$\lambda | \mu, y \sim \text{IG}\left(\frac{m-1}{2}, \frac{s^2 + m(\bar{y} - \mu)^2}{2} \right) ,$$

  where $\bar{y}$ is the sample mean and $s^2 = \sum (y_i - \bar{y})^2$

- Consider the Gibbs sampler that updates λ then μ

$$(\lambda', \mu') \to (\lambda, \mu') \to (\lambda, \mu)$$

- This sampler is geometrically ergodic
- Suppose $m = 11$, $\bar{y} = 1$, and $s^2 = 14$
- Then $E(\mu|y) = 1$ and $E(\lambda|y) = 2$
- Consider estimating $E(\mu|y)$ and $E(\lambda|y)$ with $\bar{\mu}_n$ and $\bar{\lambda}_n$

Stopped the simulation when

$$BM: \quad t_{.975,(a-1)} \frac{\hat{\sigma}_{BM}}{\sqrt{n}} + I(n < 400) < 0.04$$

$$GRD: \quad \hat{R}_{0.975} + I(n < 400) < 1.005$$

- 1000 independent replications + Starting from $\bar{y}$ for BM + Starting from draws from π for GRD - Used 4 chains for GRD

MSE	BM	GRD
MSE for $E(\mu\|y)$	3.73e-05 (1.8e-06)	0.000134 (9.2e-06)
MSE for $E(\lambda\|y)$	0.000393 (1.8e-05)	0.00165 (0.00012)

Summary

- Bayesian inference usually requires a MCMC simulation
- Metropolis-Hastings algorithm and Gibbs samplers
- Basic idea is similar to OMC, but sampling from a Markov chain yields **dependent** draws
- MCMC output analysis is often ignored or poorly understood

Lecture 18: Permutation Tests

Introduction

- Permutation tests, also called randomization tests, re-randomization tests, or exact tests
- Type of statistical significance test in which the distribution of the test statistic under the null hypothesis is obtained by calculating all possible values of the test statistic under rearrangements of the labels on the observed data points
- In other words, the method by which treatments are allocated to subjects in an experimental design is mirrored in the analysis of that design
- If the labels are exchangeable under the null hypothesis, then the resulting tests yield exact significance levels

Permutation tests

- Significance tests tell us whether an observed effect, such as a difference between two means or a correlation between two variables, could reasonably occur **just by chance** in selecting a random sample
- If not, we have evidence that the effect observed in the sample reflects an effect that is present in the population
- The general construction is as follows
  - Choose a statistic that measures the effect you are looking for
  - Construct the sampling distribution that this statistic would have if the effect were not present in the population
  - Locate the observed statistic on sampling distribution
  - A value in the main body of the distribution could easily occur just by chance
  - A value in the tail would rarely occur by chance and so is evidence that something other than chance is operating

Example: Reading

Do new **directed reading activities** improve the reading ability of elementary school students, as measured by their Degree of Reading Power (DRP) scores? A study assigns students at random to either the new method (treatment group, 21 students) or traditional teaching methods (control group, 23 students).

$$\text{statistic} = \bar{x}_{treatment} - \bar{x}_{control}$$

The null hypothesis H_0 for the resampling test is that the teaching method has no effect on the distribution of DRP scores. If H_0 is true, the DRP scores do not depend on the teaching method. Each student has a DRP score that describes that child and is the same no matter which group the child is assigned to. The observed difference in group means just reflects the accident of random assignment to the two groups.

Now we can see how to resample in a way that is consistent with the null hypothesis: imitate many repetitions of the random assignment of students to treatment and control groups, with each student always keeping his or her DRP score unchanged. Because resampling in this way scrambles the assignment of students to groups, tests based on resampling are called **permutation tests**, from the mathematical name for scrambling a collection of things.

Permutation test procedure

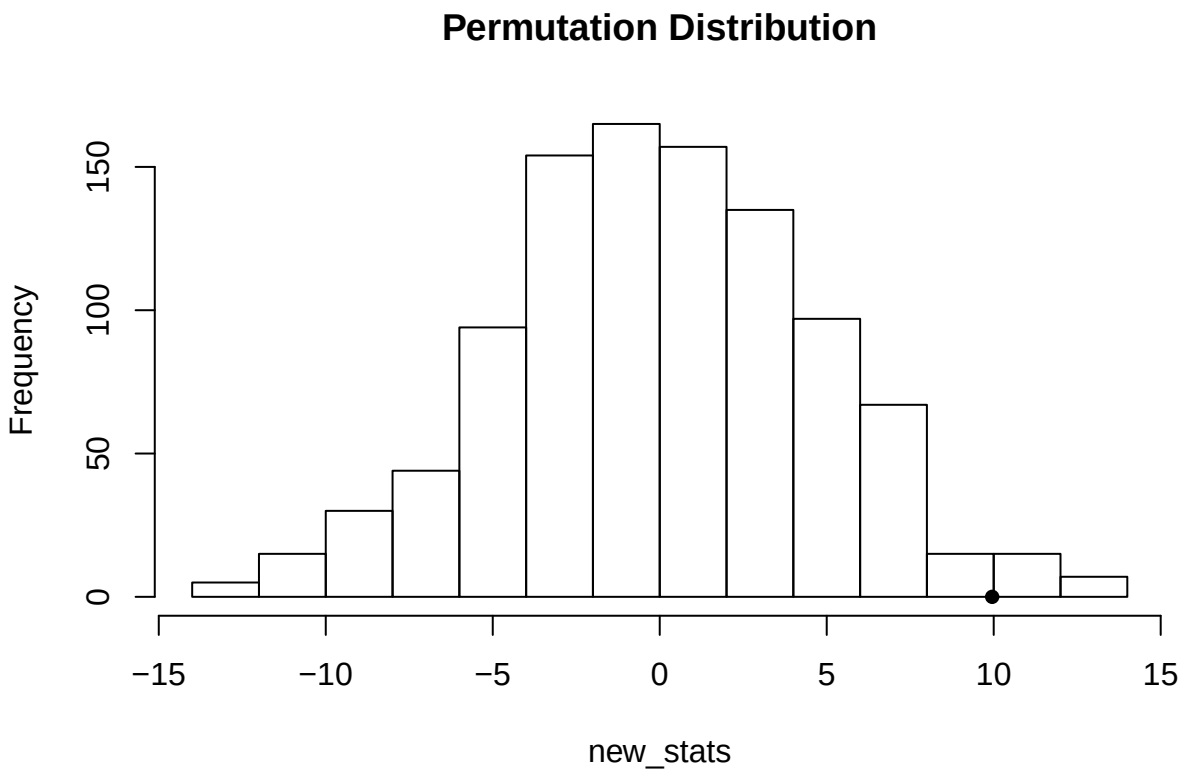
1. Choose 21 of the 44 students at random to be the treatment group; the other 23 are the control group. This is an ordinary SRS, chosen without replacement. It is called a **permutation resample**. Calculate the mean DRP score in each group, using the individual DRP scores. The difference between these means is our statistic.

2. Repeat this resampling from the 44 students hundreds of times. The distribution of the statistic from these resamples estimates the sampling distribution under the condition that H_0 is true. It is called a **permutation distribution**.
3. Compute the actually observed value of the test statistic.
4. Find the P-value.

```
T <- c(24, 43, 58, 71, 61, 44, 67, 49, 59, 52, 62, 54, 46, 43, 57, 43, 57, 56,
    53, 49, 33)
C <- c(42, 43, 55, 26, 33, 41, 19, 54, 46, 10, 17, 60, 37, 42, 55, 28, 62, 53,
    37, 42, 20, 48, 85)
n1 <- length(T)
n2 <- length(C)
Z <- c(T, C)
N <- length(Z)
obs_stat <- mean(T) - mean(C)
B <- 1000
new_stats <- numeric(B)
for (i in 1:B) {
    idx <- sample(1:N, size = n1, replace = F)
    newT <- Z[idx]
    newC <- Z[-idx]
    new_stats[i] <- mean(newT) - mean(newC)
}

pvalue <- mean(c(obs_stat, new_stats) >= obs_stat)
pvalue
```

```
## [1] 0.02297702
```

```
hist(new_stats, main = "Permutation Distribution")
points(obs_stat, 0, cex = 1, pch = 16)
```



Comparison

Comparing a **standard** t-test approach to a permutation approach brings out some general points about permutation tests versus traditional formula-based tests

- The hypotheses for a t test are stated in terms of population means

$$H_0 : \mu_{treatment} - \mu_{control} = 0$$

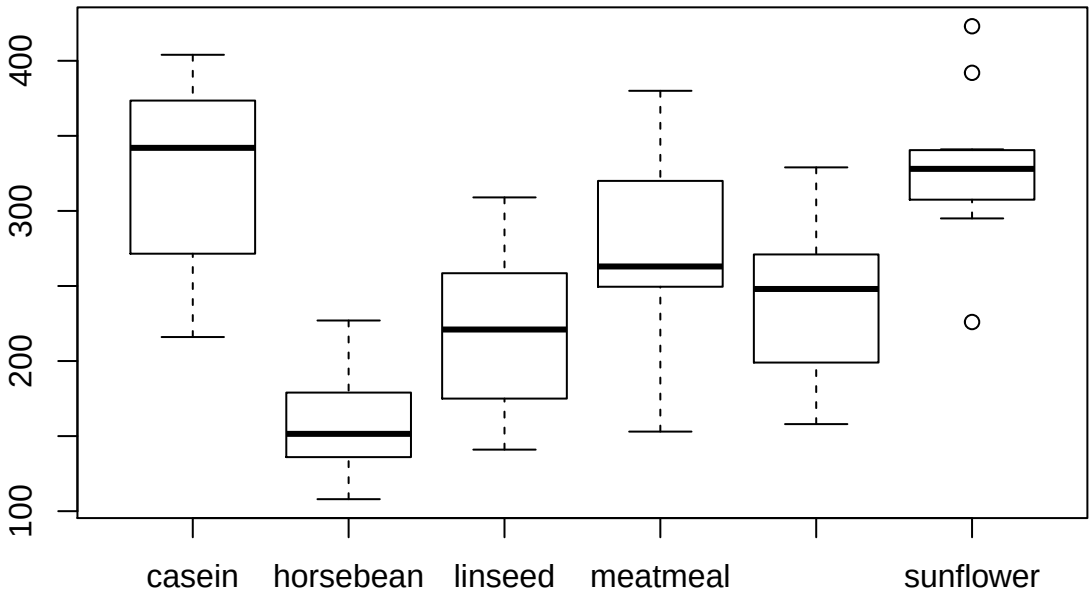
- Permutation test hypotheses are more general, i.e. the null hypothesis is **same distribution of scores in both groups**.
- The t test statistic is based on standardizing the difference in means in a clever way to get a statistic that has a t distribution under H_0. The permutation test works directly with the difference of means (or some other statistic) and estimates the sampling distribution by resampling.
- The t test gives accurate p-values when the sampling distribution of the difference of means is at least roughly normal. The permutation test gives accurate p-values even when the sampling distribution is not close to normal.

Example: `chickwts`

The permutation distribution of a statistic is illustrated for a small sample, from the data set `chickwts` in R. Weights in grams are recorded for six groups of newly hatched chicks fed different supplements.

```
data(chickwts)
attach(chickwts)
```

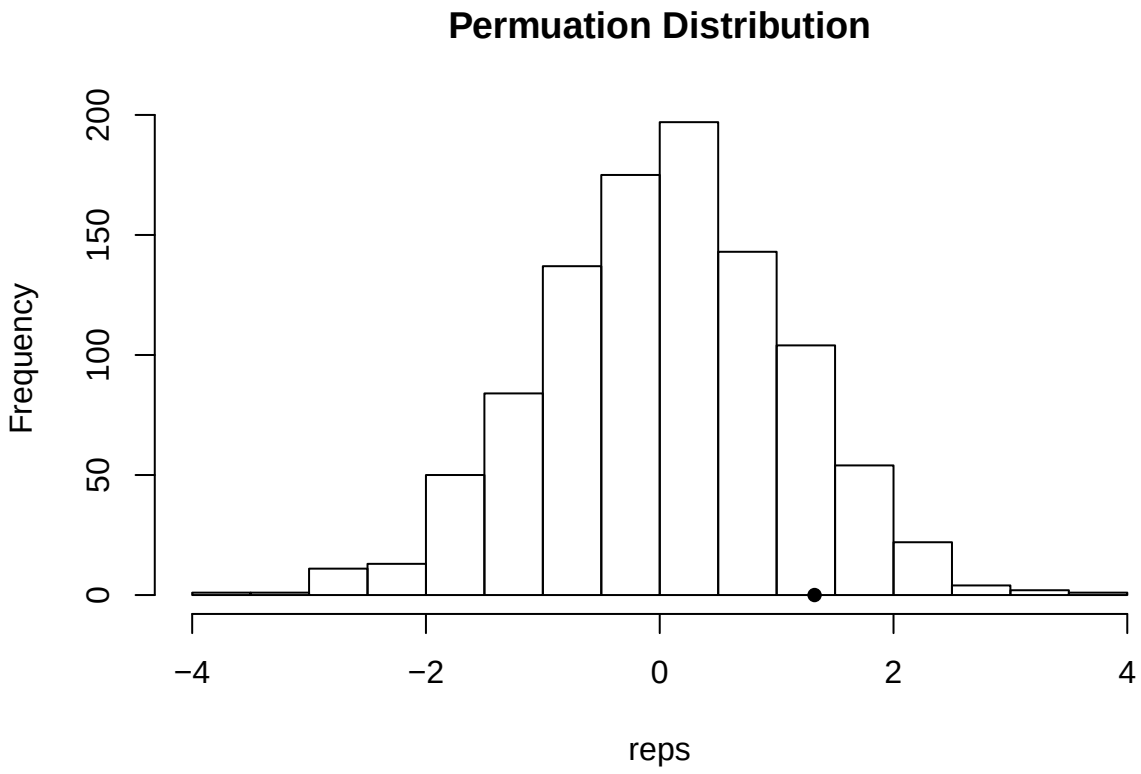
```
boxplot(formula(chickwts))
```




```
detach(chickwts)
X <- as.vector(chickwts$weight[chickwts$feed == "soybean"])
Y <- as.vector(chickwts$weight[chickwts$feed == "linseed"])

B <- 999
Z <- c(X, Y)
reps <- numeric(B)
K <- 1:26
```

```
t0 <- t.test(X, Y)$statistic
for (i in 1:B) {
    k <- sample(K, size = 14, replace = F)
    x1 <- Z[k]
    y1 <- Z[-k]
    reps[i] <- t.test(x1, y1)$statistic
}
p <- mean(c(t0, reps) >= t0)
```

```
hist(reps, main = "Permuation Distribution")
points(t0, 0, cex = 1, pch = 16)
```



Example: K-S statistic

To apply a permutation test of equal distributions, choose a test statistic that measures the difference between two distributions, for example, the two-sample Kolmogorov-Smirnov (K-S) statistic. Consider the same example as before, but now we are interested in any type of difference in the distributions of the two groups (not just the mean value as before).

The K-S statistic D is the maximum absolute difference between the ecdfs of the two samples, defined by

$$D = \sup_{1 \le i \le m+n} |F_n(z_i) - G_m(z_i)|$$

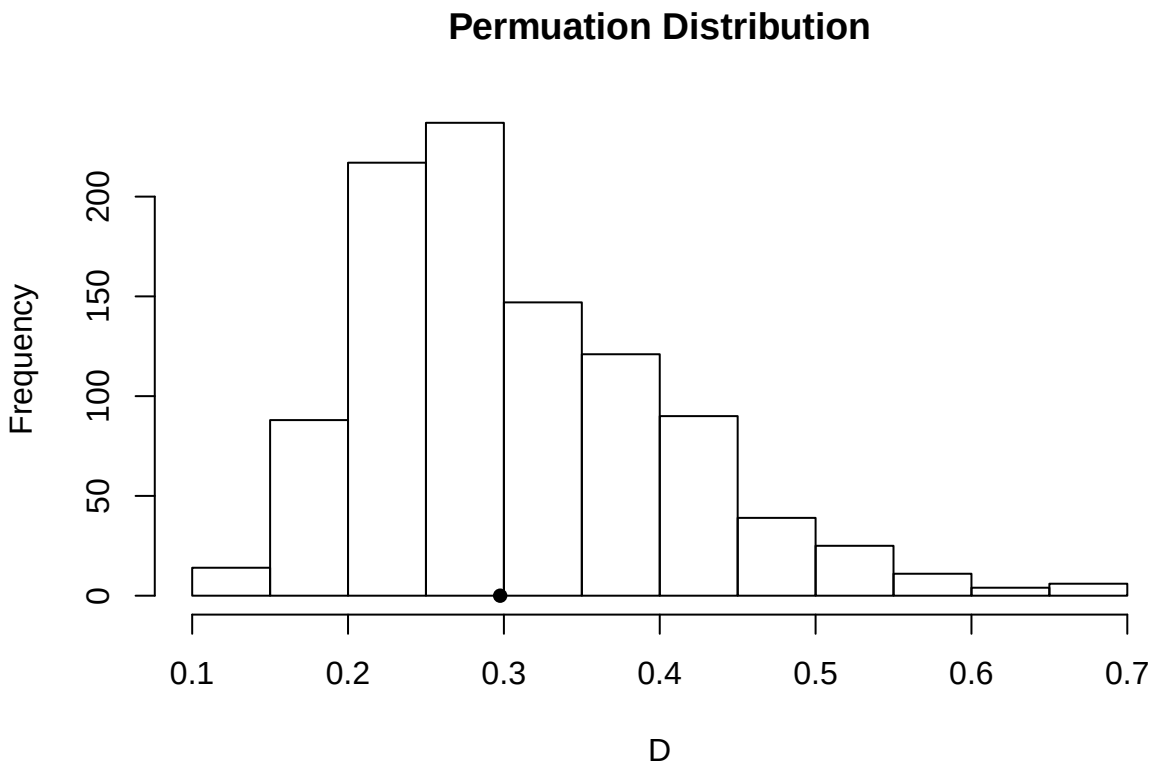
where F_n is the ecdf of the first sample $X_1, \ldots, X_n$ and G_m is the ecdf of the second sample $Y_1, \ldots, Y_m$. Note that $0 \le D \le 1$ and large values of D support the alternative $H_1 : F_X \ne F_Y$. In R, we can compute this statistic using `ks.test`.

```
D <- numeric(B)
D0 <- ks.test(X, Y, exact = F)$statistic
```

```
## Warning in ks.test(X, Y, exact = F): p-value will be approximate in the
## presence of ties
```

```
options(warn = -1)
D <- numeric(B)
for (i in 1:B) {
    k <- sample(K, size = 14, replace = F)
    x1 <- Z[k]
    y1 <- Z[-k]
    D[i] <- ks.test(x1, y1, exact = F)$statistic
}
p <- mean(c(D0, D) >= D0)
```

```
hist(D, main = "Permuation Distribution")
points(D0, 0, cex = 1, pch = 16)
```



Example: Correlation coefficients

- A study by Katz et al. (1990) asked students to answer SAT-type questions without having read the passage on which those questions were based (The SAT is a standardized exam commonly used in university admissions)
- Authors looked to see how performance on such items correlated with the SAT scores those students had when they applied to college
- Expected that those students who had the skill to isolate and reject unreasonable answers, even when they couldn't know the correct answer, would also be students who would have done well on the SAT taken sometime before they came to college

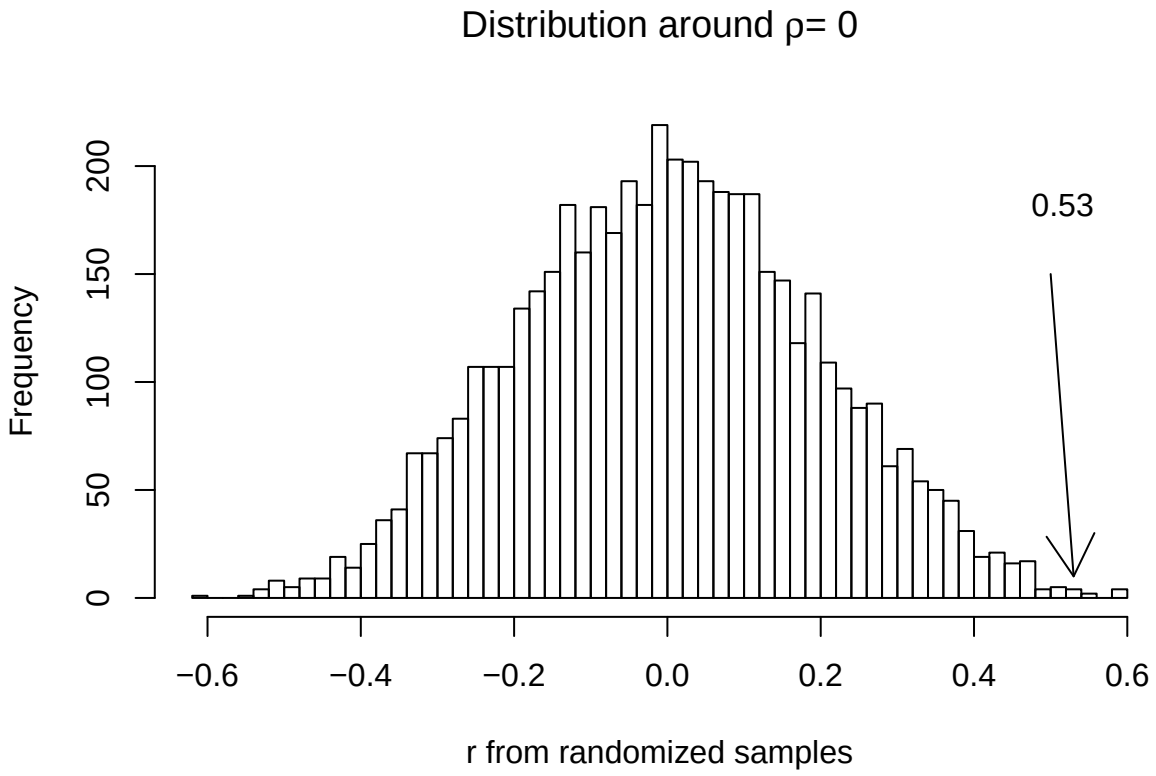
```
Score <- c(58, 48, 48, 41, 34, 43, 38, 53, 41, 60, 55, 44, 43, 49, 47, 33, 47,
    40, 46, 53, 40, 45, 39, 47, 50, 53, 46, 53)
SAT <- c(590, 590, 580, 490, 550, 580, 550, 700, 560, 690, 800, 600, 650, 580,
```

```
    660, 590, 600, 540, 610, 580, 620, 600, 560, 560, 570, 630, 510, 620)
r.obt <- cor(Score, SAT)
cat("The obtained correlation is ", r.obt, "\n")
```

```
## The obtained correlation is  0.531767
```

```
nreps <- 5000
r.random <- numeric(nreps)
for (i in 1:nreps) {
    Y <- Score
    X <- sample(SAT, 28, replace = FALSE)
    r.random[i] <- cor(X, Y)
}
prob <- length(r.random[r.random >= r.obt])/nreps
cat("Probability randomized r >= r.obt", prob)
```

```
## Probability randomized r >= r.obt 0.0016
```



Bootstrap versus randomization

- When we bootstrap for correlations, we keep x_i and y_i pairs together, and randomly sample pairs of scores with replacement. That means that if one pair is 45 and 360, we will always have 45 and 360 occur together, often more than once, or neither of them will occur. What this means is that the expectation of the correlation between X and Y for any resampling will be the correlation in the original data.
- When we use a randomization approach, we permute the Y values, while holding the X values constant. For example, if the original data were

```
x <- c(45, 53, 73, 80)
y <- c(22, 30, 29, 38)
```

then two resamples might be

```
rbind(x, sample(y, size = 4, replace = F))
```

```
##   [,1] [,2] [,3] [,4]
## x   45   53   73   80
##     29   30   22   38
```

```
rbind(x, sample(y, size = 4, replace = F))
```

```
##   [,1] [,2] [,3] [,4]
## x   45   53   73   80
##     30   29   22   38
```

- Notice the top row always stays in the same order, while the bottom row is permuted randomly. This means that the expected value of the correlation between X and Y will be 0.00, not the correlation in the original sample.
- Helps to explain why bootstrapping focuses on confidence limits around ρ, whereas the randomization procedure focuses on confidence limits around 0.

Summary

- Fisher's exact test is a commonly used permutation test for evaluating the association between two dichotomous variables
  - `fisher.test` in R can simulate a p-value via Monte Carlo
  - `XNomial` for larger tables
- More on Randomization Tests
- Don't always need to build our own permutation distributions, in some cases can use `coin` R package

Lecture 19: Databases

This is a basic intro to databases, if you know about it, I would recommend to skip it.

Agenda

- Thanks to Prof. Cosma Shalizi (CMU Statistics) for this material
- What databases are, and why
- SQL
- Interfacing R and SQL

Databases

- A **record** is a collection of **fields**
- A **table** is a collection of records which all have the same fields (with different values)
- A **database** is a collection of tables

Databases vs. Dataframes

- R's dataframes are actually tables

R jargon	Database jargon
column	field
row	record
dataframe	table
types of the columns	table **schema**
bunch of related dataframes	database

Why Do We Need Database Software?

- *Size*
  - R keeps its dataframes in memory
  - Industrial databases can be much bigger
  - Work with selected subsets
- *Speed*
  - Clever people have worked very hard on getting just what you want fast
- *Concurrency*
  - Many users accessing the same database simultaneously
  - Lots of potential for trouble (two users want to change the same record at once)

The Client-Server Model

- Databases live on a **server**, which manages them
- Users interact with the server through a **client** program
- Lets multiple users access the same database simultaneously

SQL

- **SQL** (**structured query language**) is the standard for database software
- Mostly about **queries**, which are like doing a selection in R

```
debt[debt$Country=="France",c("growth","ratio")]
with(debt,debt[Country=="France",c("growth","ratio")])
subset(x=debt,subset=(Country=="France"),select=c("growth","ratio"))
```

- Let's look at how SQL does stuff like this

SELECT

```
SELECT columns or computations
  FROM table
  WHERE condition
  GROUP BY columns
  HAVING condition
  ORDER BY column [ASC|DESC]
  LIMIT offset,count;
```

- `SELECT` is the first word of a query, then modifiers say which fields/columns to use, and what conditions records/rows must meet, from which tables
- The final semi-colon is obligatory

```
SELECT PlayerID,yearID,AB,H FROM Batting;
```

Four columns from table `Batting`

```
SELECT * FROM Salaries;
```

All columns from table `Salaries`

```
SELECT * FROM Salaries ORDER BY Salary;
```

As above, but by ascending value of `Salary`

```
SELECT * FROM Salaries ORDER BY Salary DESC;
```

Descending order

```
SELECT * FROM Salaries ORDER BY Salary DESC LIMIT 10;
```

top 10 salaries

Picking out rows meeting a condition

```
SELECT PlayerID,yearID,AB,H FROM Batting WHERE AB > 100 AND H > 0;
```

vs.

```
Batting[Batting$AB>100 & Batting$H > 0, c("PlayerID","yearID","AB","H")]
```

Calculated Columns

- SQL knows about some simple summary statistics:

```
SELECT MIN(AB), AVG(AB), MAX(AB) FROM Batting;
```

- It can do arithmetic

```
SELECT AB,H,H/CAST(AB AS REAL) FROM Batting;
```

Because `AB` and `H` are integers, and it won't give you a fractional part by default

- Calculated columns can get names:

```
SELECT PlayerID,yearID,H/CAST(AB AS REAL) AS BattingAvg FROM Batting
  ORDER BY BattingAvg DESC LIMIT 10;
```

Aggregating

We can do calculations on value-grouped subsets, like in `aggregate` or `d*ply`

```
SELECT playerID, SUM(salary) FROM Salaries GROUP BY playerID
```

Selecting Again

- First cut of records is with `WHERE`
- Aggregation of recordw with `GROUP BY`
- Post-aggregation selection with `HAVING`

```
SELECT playerID, SUM(salary) AS totalSalary FROM Salaries GROUP BY playerID
  HAVING totalSalary > 200000000
```

JOIN

- So far `FROM` has just been one table
- Sometimes we need to combine information from many tables

patient_last	patient_first	physician_id	complaint
Morgan	Dexter	37010	insomnia
Soprano	Anthony	79676	malaise
Swearengen	Albert	NA	healthy as a goddam horse
Garrett	Alma	90091	nerves
Holmes	Sherlock	43675	nicotine-patch addiction

physician_last	physician_first	physicianID	plan
Meridian	Emmett	37010	UPMC
Melfi	Jennifer	79676	BCBS
Cochran	Amos	90091	UPMC
Watson	John	43675	VA

- Suppose we want to know which doctors are treating patients for insomnia
- Complaints are in one table
- Physicians are in the other
- In R, we'd use `merge` to link the tables up by `physicianID`
- Here, `physician_id` or `physicianID` is acting as the **key** or **unique identifier**
- SQL doesn't have `merge`, it has `JOIN` as a modifier to `FROM`

```
SELECT physician_first, physician_last FROM patients INNER JOIN physicians ON patients.physician_id == p
```

Creates a (virtual) table linking records where `physician_id` in one table matches `physicianID` in the other

- If the names were the same in the two tables, we could write (e.g.)

```
SELECT nameLast,nameFirst,yearID,AB,H FROM Master INNER JOIN Batting
  USING(playerID);
```

`INNER JOIN ... USING` links records with the same value of `playerID`

- There are some syntax variants here; see the handout
- `LEFT OUTER JOIN` includes records from the first table which don't match any record in the 2nd
  - The "extra" records get `NA` in the 2nd table's fields
- `RIGHT OUTER JOIN` is just what you'd think
  - so is `FULL OUTER JOIN`

Updated Translation Table

R jargon	Database jargon
column	field
row	record
dataframe	table
types of the columns	table schema
bunch of dataframes	database
selections, `subset`	`SELECT ... FROM ... WHERE ... HAVING`
`aggregate`, `d*ply`	`GROUP BY`
`merge`	`JOIN`
`order`	`ORDER BY`

Connecting R to SQL

- SQL is a language; database management systems (DMBS) actually implement it and do the work
  - MySQL, SQLite, etc., etc.
- They all have somewhat different conventions
- The R package `DBI` is a unified interface to them
- Need a separate "driver" for each DBMS

```
install.packages("DBI", dependencies = TRUE) # Install DBI
install.packages("RSQLite", dependencies = TRUE) # Install driver for SQLite
library(RSQLite)
drv <- dbDriver('SQLite')
con <- dbConnect(drv, dbname="baseball.db")
```

`con` is now a persistent connection to the database `baseball.db`

```
dbListTables(con)         # Get tables in the database (returns vector)
dbListFields(con, name)  # List fields in a table
dbReadTable(con, name)   # Import a table as a data frame
```

```
dbGetQuery(conn, statement)
df <- dbGetQuery(con, paste(
  "SELECT nameLast,nameFirst,yearID,salary",
  "FROM Master NATURAL JOIN Salaries"))
```

Usual workflow:

- Load the driver, connect to the right database
- R sends an SQL query to the DBMS
- SQL **executes the query**, sending back a manageably small dataframe
- R does the actual statistics
- Close the connection when you're done

Going the Other Way

- The `sqldf` package lets you use SQL commands on dataframes
- Mostly useful if you already know SQL better than R...

Summary

- A database is basically a way of dealing efficiently with lots of potentially huge dataframes
- SQL is the standard language for telling databases what to do, especially what queries to run
- Everything in an SQL query is something we've practiced already in R
  - subsetting/selection, aggregation, merging, ordering
- Connect R to the database, send it an SQL query, analyse the returned dataframe
- More information at [http://www.stat.cmu.edu/~cshalizi/statcomp/14/]