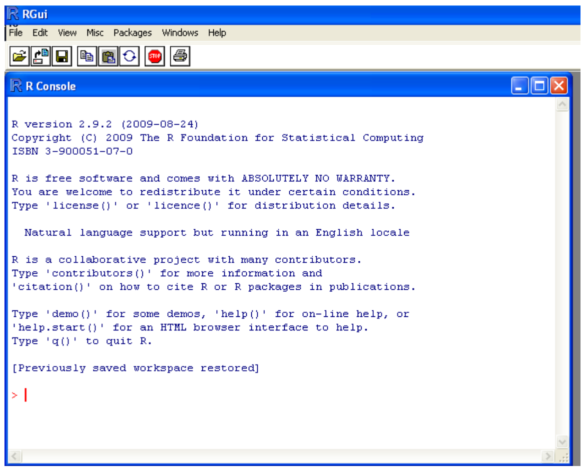
Getting started with R

This handout gives a very brief introduction to R to give you a rough idea of some basic R features. We assume that R has been installed on your computer.

Start up R

Start R by double-clicking on the R shortcut on the desktop. A window called the R **Console** will open up.



Type commands after the prompt > and then press the <**ENTER**> key.

Note: anything after the pound symbol # is a comment—explanatory text that is not executed.

```
> 4*9           # Simple arithmetic
[1] 36
```

If you strike the <ENTER> key before typing a complete expression, you will see the *continuation prompt*, the plus sign (+). For example, suppose you wish to calculate $3+2*(8-4)$, but you accidentally strike the <ENTER> key after typing the 8:

```
> 3+2*(8  <ENTER>
+
```

Finish the expression by typing -4) after the +

```
+  - 4) <ENTER>
[1] 11
```

Create a sequence incrementing by 1:

```
> 20:30
[1] 20 21 22 23 24 25 26 27 28 29 30
```

We will create an object called `dog` and assign it the values 1, 2, 3, 4, 5. The symbol <- is the assignment operator.

```
> dog <- 1:5
> dog
[1] 1 2 3 4 5
> dog + 10
[1] 11 12 13 14 15
> 3*dog
[1] 3 6 9 12 15
> sum(dog)        # 1+2+3+4+5
[1] 15
```

The object `dog` is called a *vector*.

If you need to abort a command, press the escape key <**ESC**>. The up arrow key ↑ can be used to recall previous entries.

To obtain help on any of the commands, type the name of the command you wish help on:

```
> ?sum
```

Importing data

Data for the textbook can be downloaded from the web site
`https://sites.google.com/site/ChiharaHesterberg`

For instance, let's start with the Flight Delays Case Study (see Chapter 2, **Exploratory Data Analysis**, of the text for a description of this data set). For now, we assume the file `FlightDelays.csv` is on the desktop of your computer. We use the `read.csv` command to read it into our `R` session,

```
> FlightDelays <- read.csv("C:/Users/UserName/Desktop/FlightDelays.csv")
```

or on the Macintosh

```
> FlightDelays <- read.csv("Users/UserName/Desktop/FlightDelays.csv")
```

where `UserName` is the name used to login to this session.

To view the names of the variables in `FlightDelays`:

```
> names(FlightDelays)
```

To view the first part of the data, use the `head` command:

```
> head(FlightDelays)
```

(What do you suppose the `tail` command does?)

The columns are the *variables*. There are two types of variables: *numeric*, for example, `FlightLength` and `Delay` and *factor* (also called *categorical*) (for example `Carrier` and `DepartTime`). The rows are called *observations* or *cases*.

To check the size (dimension) of the data frame, type

```
> dim(FlightDelays)
[1] 4029  10
```

This tells us that there are 4029 observations and 10 columns.

Tables, bar charts and histograms

The factor variable `Carrier` in the **FlightDelays** data set assigns each flight to one of two *levels*: UA or AA. To obtain the summary of this variable

```
> table(FlightDelays$Carrier)
  AA   UA
2906 1123
```

Remark: `R` is *case-sensitive*! `Carrier` and `carrier` are considered different.

To visualize the distribution of a factor variable, create a *bar chart*:

```
> barplot(table(FlightDelays$Carrier))
```

To compare two categorical variables, the airline (`Carrier`) and whether or not the flight was delayed by more than 30 minutes (`Delayed30`):

```
> table(FlightDelays$Carrier, FlightDelays$Delayed30)

         Delayed30
Carrier   No  Yes
     AA 2513  393
     UA  919  204
```

To see the distribution of a numeric variable, create a histogram.

```
> hist(FlightDelays$Delay)
```

The shape of the distribution of this variable is *right-skewed*.

Numeric Summaries

Because it is a bit cumbersome to use the syntax `FlightDelays$Delay` each time we want to work with the `Delay` variable, we will create a new object `delay`:

```
> delay <- FlightDelays$Delay
```

```
> mean(delay)
> median(delay)
```

To compute the *trimmed* mean, trimming by 25% on either side,

```
> mean(delay, trim = .25)
```

Other basic statistics:

```
> max(delay)
> min(delay)
> range(delay)
> var(delay)          #variance
> sd(delay)           #standard deviation
> quantile(delay)     #quartiles
```

If you need the population variance (that is, denominator of 1/n instead of 1/(n-1)),

```
> n <- length(delay)
> (n-1)/n*var(delay)
```

The `tapply` command

The `tapply` command allows you to compute numeric summaries on values based on levels of a factor variable. For instance, find the mean flight delay length by carrier,

```
> tapply(delay, FlightDelays$Carrier, mean)
```

or the median flight delay length by time of departure

```
> tapply(delay, FlightDelays$DepartTime, median)
```

Boxplots

Boxplots give a visualization of the 5-number summary of a variable.

```
> summary(delay)        #numeric summary
> boxplot(delay)
```

To compare the distribution of a numeric variable across the levels of a factor variable

```
> tapply(delay, FlightDelays$Day, summary)
> boxplot(Delay ~ Day, data = FlightDelays)
> tapply(delay, FlightDelays$DepartTime, summary)
> boxplot(Delay ~ DepartTime, data = FlightDelays)
```

The `boxplot` command gives the option of using a formula syntax, `Numeric` $\sim$ `Factor`. In this case, you do not need to use the $ syntax if you specify the data set.

Misc. Remarks

- Functions in `R` are called by typing their name followed by arguments surrounded by *parentheses*: ex. `hist(delay)`. Typing a function name without parentheses will give the code for the function.

  ```
  > sd
  ```

- We saw earlier that we can assign names to data (we created a vector called `dog`.) Names can be any length, must start with a letter, and may contain letters or numbers:

  ```
  > fish25 <- 10:35
  > fish25
  ```

  Certain names are **reserved** so be careful to not use them: `cat`, `c`, `t`, `T`, `F`,...

  To be safe, before making an assignment, type the name:

  ```
  > whale
  [1] Problem: Object "whale" not found
  ```

  Safe to use `whale`!

  ```
  > whale <- 200
  > objects()
  > rm(whale)
  > objects()
  ```

- In general, `R` is space-insensitive.

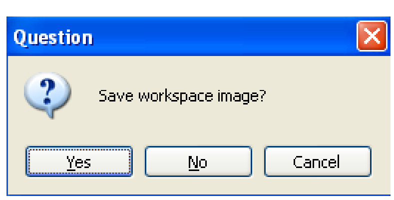
  ```
  > 3 +4
  > 3+   4
  > mean(3+5)
  > mean ( 3 + 5 )
  ```

  BUT, the assignment operator must not have spaces! `<-` is different from `<  -`

- To quit, type

  ```
  > q()
  ```

  You will be given an option to **Save the workspace image**.



Select **Yes**: all objects created in this session are saved to your working directory so that the next time you start up `R`, if you load this working directory, these objects will still be available. You will not have to re-import `FlightDelays`, for instance, nor recreate `delay`.

You can, for back-up purposes, save data to an external file/disk by using, for instance, the `write.csv` command. See the help file for more information.

Workspace Management

It will be convenient to set up a folder to store the data for this textbook. In the directory where you keep your document, create a folder called `R` and then a subfolder called **MathStats**. The full path depending on the operating system (PC or Mac) will look something like

C:\Users\UserName\Documents\R\MathStats,

or

Users\UserName\Documents\R\MathStats.

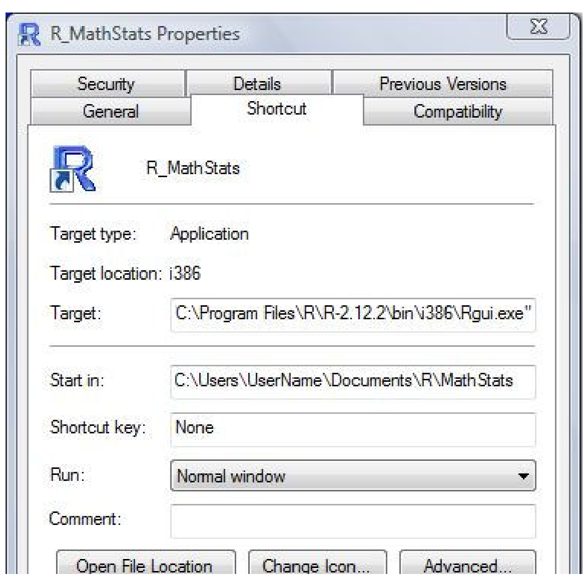
(the exact path will depend on your computer's particular configuration).

Next, we will set up `R` to start in the above directory.

Windows

On the desktop of your PC, find the `R` shortcut, right-click and from the context menu, select **Properties**. In the **Start In:** field, type the path to this folder, for instance

`C:\Users\UserName\Documents\R\MathStats`



and then click **OK**.

Starting R by clicking on this short-cut will automatically load the **MathStats** working directory. To check this,

```
> getwd()
[1] "C:/Users/UserName/Documents/R/MathStats"
```

Copy data files to the working directory. To read them into your R session, you then only need to type the name of the file, rather than a long path. For instance, to read in the data for the Flight Delays Case Study,

```
> FlightDelays <- read.csv("FlightDelays.csv")
```

Rename your shortcut (say to **R.MathStats**) since you may want to set up different working directories for different projects and classes.

Macintosh

On the Macintosh, dragging the **MathStats** folder icon onto the R icon will start R in the **MathStats** working directory.

Alternatively, if R is already running, then at the menu, under **R** > **Preferences**, click on **Start-up**. You can change the default working directory to be the **MathStats** folder. Then R will always start with the **MathStats** being the working directory.

Copy data files to the working directory. To read them into your R session, you then only need to type the name of the file, rather than a long path. For instance, to read in the data for the Flight Delays Case Study,

```
> FlightDelays <- read.csv("FlightDelays.csv")
```

RStudio

RStudio is an integrated development environment for R. It is free and can be obtained from `http://www.rstudio.org`. **RStudio** runs on Windows, Macs and Linux. It has a nice interface for managing different projects and workspaces.

Vectors in R

The basic data object in R is the vector. Even scalars are vectors of length 1.

There are several ways to create vectors.

The `:` operator creates sequences incrementing/decrementing by 1.

```
> 1:10
> 5:-3
```

The `seq()` function creates sequences also.

```
> seq(0, 3, by = .2)
> seq(0, 3, length = 15)
> quantile(delay, seq(0, 1, by = .1))     #deciles of delay
```

To create vectors with no particular pattern, use the `c()` function (`c` for **c**ombine).

```
> c(1, 4, 8, 2, 9)
> x <- c(2, 0, -4)
> x
> c(x, 0:5, x)
```

For vectors of characters,

```
> c("a", "b", "c", "d")
```

or logical values (note that there are no double quotes):

```
> c(T, F, F, T, T, F)
```

The `rep()` command for repeating values:

```
> rep("a", 5)
> rep(c("a", "b"), 5)
> rep(c("a", "b"), c(5, 2))
```

The "class" attribute

Use `data.class` to determine the class attribute of an object.

```
> state.name
> data.class(state.name)
> state.name == "Idaho"
> data.class(state.name == "Idaho")
> head(FlightDelays$Carrier)
> data.class(FlightDelays$Carrier)
```

Basic Arithmetic

```
> x <- 1:5
> x - 3
> x*10
> x/10
> x^2
> 2^x
> log(x)

> w <- 6:10
> w
> x*w                  #coordinate-wise multiplication
```

Logical expressions

```
> x < 3
> x == 4
```

Subsetting

In many cases, we will want only a portion of a data set. For subsetting a vector, the basic syntax is `vector[`*index*`]`. In particular, note the use of *brackets* to indicate that we are subsetting.

```
> z <- c(8, 3, 0, 9, 9, 2, 1, 3)
```

The fourth element of `z`:

```
> z[4]
```

The first, third and fourth element,

```
> z[c(1, 3, 4)]
```

All elements *except* the first, third and fourth:

```
> z[-c(1, 3, 4)]
```

To return the values of `z` less than 4, we first introduce the `which` command:

```
> which(z < 4)            # which positions are z values < 4?
> index <- which(z < 4)   # store in index
> z[index]                # return z[c(2, 3, 6, 7)]
```

The `subset` command

The `subset` command is used to extract rows of the data that satisfy certain conditions.

If necessary, re-import the `FlightDelays` data set.

Recall that we extracted the variable `delay` using the $ syntax:

```
> delay <- FlightDelays$Delay
```

The `subset` command can also be used:

```
> delay <- subset(FlightDelays, select = Delay, drop = TRUE)
```

The `select =` argument indicates which column to choose.

The `drop = TRUE` argument is need to create a `vector`. Compare the output of

```
> subset(FlightDelays, select = Delay, drop = TRUE)
```

to

```
> subset(FlightDelays, select = Delay)
```

The second output (omitting the `drop = TRUE` argument) is a `data.frame` object (more on data frames later).

Suppose you wish to extract the flight delay lengths for all days except Monday and then find the mean delay length:

```
> delay2 <- subset(FlightDelays, select = Delay, subset = Day != "Mon",
  drop = TRUE)
> mean(delay2)
```

The `subset =` argument is a logical expression indicating which rows to keep. We want those days not equal to Monday.

To extract the delay lengths for Saturday and Sunday only

```
> delay3 <- subset(FlightDelays, select = Delay,
  subset = (Day == "Sat" | Day == "Sun"), drop = TRUE)
> mean(delay3)
```

The vertical bar | stands for "or."

Suppose you want to find those observations when the delay length was greater than the mean delay length. We'll store this in a vector called `index`.

```
> index <- which(delay > mean(delay))
> head(index)
[1]  2 10 12 14 15 16
```

Thus, observations in rows 2, 10, 12, 14, 15, 16 are the first six that correspond to flights that had delays that were larger than the average delay length.

Vectorized operators

==	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
&	vectorized AND
|	vectorized OR
!	not

Programming Note: The vectorized AND and OR are for use with vectors (when you are extracting subsets of vectors). For control in programming (ex. when writing `for` or `if` statements), the operators are && and ||.

Misc. commands on vectors

```
> length(z)        # number of elements in z
> sum(z)           # add elements in z
> sort(z)          # sort in increasing order
```

The `sample` command is used to obtain random samples from a set.

For instance, to permute the numbers $1, 2, \cdots, 10$:

```
> sample(10)
```

To obtain a random sample from $1, 2, \cdots, 10$ of size 4 (without replacement):

```
> sample(10, 4)
```

Without replacement is the default; if you want to sample with replacement:

```
> sample(10, 4, replace = TRUE)
```

The built-in vector `state.name` contains the 50 states.

```
> state.name
```

Draw a random sample of 20 states:

```
> sample(state.name, 20)
```

If you want to sample with replacement,

```
> sample(state.name, 20, replace=TRUE)
```

Suppose you wish to create two vectors, one containing a random sample of 20 of the states, the other vector containing the remaining 30 states.

We obtain a random sample of size 20 from 1, 2, . . . , 50 and store in the vector `index`:

```
> index <- sample(50, 20)
```

See what this sample looks like:

```
> index
```

`tempA` will contain the random sample of 20 states:

```
> tempA <- state.name[index]
```

`tempB` will contain the remaining 30 states. Recall that in subsetting, the negative of a number means to exclude that observation.

```
> tempB <- state.name[-index]

> tempA

> tempB
```

Data Frames in R

Most data will be stored in data frames, rectangular arrays which usually are formed by combining columns of vectors. `FlightDelays` is an example of a data frame.

```
> data.class(FlightDelays)
```

Subsetting data frames

For subsetting a data frame, use the syntax
`data[row.index, column.index]`.

For instance, row 5, column 3:

```
> FlightDelays[5, 3]
```

or rows 1 through 10, columns 1 and 3:

```
> FlightDelays[1:10, c(1, 3)]
```

or all rows *except* 1 through 10, and keep columns 1 and 3:

```
> FlightDelays[-(1:10), c(1, 3)]
```

To extract *all* rows, but columns 1 and 3

```
> FlightDelays[, c(1, 3)]
```

and rows 1:100 and *all* columns

```
> FlightDelays[1:100, ]
```

Use the `subset` command to extract rows based on some logical condition.

Create a subset of just the Tuesday data:

```
> DelaysTue <- subset(FlightDelays, subset = Day == "Tue")
> head(DelaysTue)
```

Create a subset of just the Tuesday data and columns $1, 6, 7$:

```
> DelaysTue <- subset(FlightDelays, Day == "Tue", select = c(1, 6, 7))
> head(DelaysTue)
```

Plots in R

There are three basic plotting functions in R: high-level plots, low-level plots, and the layout command `par`. Basically, a high-level plot function creates a complete plot and a low-level plot function adds to an existing plot, that is, one created by a high-level plot command.

High-Level Plot Functions

Some of the basic plot functions include:

Function	**Description**
`plot`	scatter/line plot
`hist`	histogram
`barplot`	barplot
`boxplot`	boxplot
`qqnorm`	normal-quantile

Download the example data set **States03** from `http://apps.carleton.edu/curricular/math/resources/rcomputing/`, then import it into your session.

If you are using `RStudio`, then at the menu, **Tools** > **Import Dataset** > **From Text File...** and navigate to the location of the file.

Or at the command line, use `read.csv` to import the data:

```
> States03 <- read.csv("States03.csv")
```

(exact path will vary depending on where you saved the file).

Basic single variable plots:

```
> barplot(table(States03$Region))
> hist(States03$Poverty)
```

To create a scatter plot, there are two approaches:

```
> plot(States03$Unemp, States03$Poverty, xlab = "Unemployment", ylab = "Poverty")
> plot(Poverty ~ Unemp, data = States03, xlab = "Unemployment", ylab = "Poverty")
```

In the first approach, provide the `plot` command with the x-variable, then the y-variable. In the second approach, if the data are contained in a data frame, then provide the names of the variable `Y` ∼ `X` along with the name of the data frame.

High-level functions may also take optional arguments that enhance the plot.

```
> hist(States03$Poverty, main = "Poverty", xlab = "percent",
       xlim = c(0, 24), ylim = c(0, 20))
> plot(1:19, 1:19, pch = 1:19, xlab = "x", ylab = "y")
> pie(rep(1, 8), col = 1:8)
```

Option	Description
pch	point character (pch=1, 2, ...)
lty	line type (lty=1, 2, ...)
lwd	line thickness (lwd= 1, 2,...)
col	color (col="red", "blue",...)
xlim	x-axis limits: xlim=c(min,max)
ylim	y-axis limits
xlab	x-axis label: xlab="my label"
ylab	y-axis label
main	main title
sub	sub title

To plot smooth curves, use the `curve` command. The first argument must be an expression in terms of x:

```
> curve(x^2, from = 0, to = 2)
> curve(cos(x), from = 0, to = pi)
> curve(cos(x), from = 0, to = pi, lty = 4, col = "red")
```

Low-level Plot Functions

Low-level plot functions can be executed only after a high-level plot has been created. For example,

```
> plot(Poverty ~ Unemp, data = States03, xlab = "Unemployment", ylab = "Poverty")
> abline(v = mean(States03$Unemp), lty = 2)  #vertical line at mean unemployment rate,
> text(30, 18, "mean unemployment rate")   #text at (30, 18)
> title("Data from 2003")
```

The `abline` function has several options:

`abline(3, 5)` adds the straight line $y = 3 + 5x$

`abline(v = 2)` adds the vertical line, $x = 2$

`abline(h = 0)` adds the horizontal line, $y = 0$

```
> plot(Poverty ~ ColGrad, data = States03, col = "blue", pch = 19, xlab = "College grad (%)",
   ylab = "Poverty (%)")
> points(Uninsured ~ ColGrad, data = States03, col = "red", pch = 19)
> mtext("Percent uninsured", side = 4)
> legend("bottomleft", legend = c("Y: Poverty","Y: Uninsured"), col = c("blue","red"),
  pch = c(16, 16))
```

You can also use different plotting symbols for different levels of a factor variable:

Probability Distributions in R

Continuous

Distributions	**root**
beta	`beta`
Cauchy	`cauchy`
chi-square	`chisq`
exponential	`exp`
F	`f`
gamma	`gamma`
normal	`norm`
student's t	`t`
uniform	`unif`
Weibull	`weibull`

In the continuous case, `d`*root* returns the density, `p`*root* a cumulative probability, `q`*root* a quantile, `r`*root* a random number.

Probability

If X follows $N(0, 1)$, then to find $P(X \leq 1.25) = \Phi(1.25)$, that is, the amount of area under the standard normal density curve to the left of $x = 1.25$,

```
> pnorm(1.25)
```

By default, the `norm` function assumes $\mu = 0, \sigma = 1$ (that is, you are working with the standard normal distribution). For other means and standard deviations, specify them in the argument. For example, if $X \sim N(\mu = 2, \sigma = 3)$, then to find $F(2.8) = P(X \leq 2.8)$,

```
> pnorm(2.8, 2, 3)
```

If X follows a chi-square distribution with 25 degrees of freedom then to compute $F(13.9) = P(X \leq 13.9)$,

```
> pchisq(13.9, 25)
```

If X follows an exponential distribution with parameter $\lambda = 10$, then to compute $P(X > 4)$,

```
>  1 - pexp(4, 10)
```

or

```
> pexp(4, 10, lower.tail=FALSE)
```

If T follows a t-distribution with 7 degrees of freedom, then to find the probability that $T \leq 3.9$, type

```
> pt(3.9, 7) # pt(t-value, d.f)
```

Quantiles

To find the 25^{th} percentile, that is, the value q such that $P(X \leq q) = .25$ for X from $N(0,1)$,

```
> qnorm(.25)
[1] -0.6744898
```

In other words, the amount of area under the pdf to the left of $x = -0.6744$ is 0.25. Or, if F denotes the cdf of the distribution, then $F^{-1}(0.25) = -0.6744$.

The .75 quantile for $N(2,3)$ can be found by

```
> qnorm(.75, 2,3)
[1] 4.023469
```

In other words, the amount of area under the density curve and to the left of $x = 4.023469$ is .75, or if F denotes the cdf, then $F^{-1}(0.75) = 4.023469$.
For T from a t-distribution with 13 degrees of freedom, to find value t such that $P(T > t) = .025$, which is equivalent to $F(t) = P(T \leq t) = 0.975$, type

```
> qt(.975, 13)
```

Random numbers

To generate 100 random numbers from the normal distribution $N(0,1)$, type

```
> rnorm(100)
> x <- rnorm(100)
> hist(x)
```

Ten random numbers from the chi-square distribution with 23 degrees of freedom,

```
> rchisq(10, 23)
```

Plotting the density curve (pdf)

To plot the pdf for $N(0,1)$ for $-3 \leq x \leq 3$, use the `curve` function with the pdf `dnorm` provided as an argument.

```
> curve(dnorm(x), from = -3, to = -3)
> w <- rnorm(50)                        # random sample from N(0,1)
> hist(w, freq = TRUE)                  # scale to area 1
> curve(dnorm(x), add = TRUE)           # impose normal density
```

```
> hist(w, freq = TRUE, ylim = c(0, .5)) # widen y-axis range
> curve(dnorm(x), add=TRUE)
```

To plot the pdf for the chi-square distribution with 14 degrees of freedom,

```
> curve(dchisq(x, 14), from = 0, to = 20)
```

Discrete

Distribution	**root**
binomial	`binom`
geometric	`geom`
hypergeometric	`hyper`
negative binomial	`nbinom`
Poisson	`pois`

Preface each of the above roots with either `d, p, q` or `r`.
`d`*root* is the probability mass function so returns a probability, `p`*root* returns a cumulative probability (cmf), and `q`*root* returns a quantile, and `r`*root* returns a random number.
The quantile function is the inverse of the CDF, $F(t) = P(X \leq t) = \sum_{k \leq t} P(X = k)$.

Example Binomial

Suppose you have a biased coin that has a probability of 0.8 of coming up heads.
The probability of getting 5 heads in 16 tosses of this coin is

```
> dbinom(5,16, .8)
```

Check this answer by calculating directly
$\binom{16}{5}.8^5 \cdot .2^{11}$,

```
> choose(16,5)*.8^5*.2^11
```

The probability of getting at most 5 heads in 16 tosses is

```
> pbinom(5,16,.8)
```

In other words, `pbinom(5, 16, .8)` is computing: `dbinom(0,16,.8)+dbinom(1,16,.8)`
`+dbinom(2,16,.8)+dbinom(3,16,.8)`
`+dbinom(4,16,.8)+dbinom(5,16,.8)`

The 0.25 quantile is

```
> qbinom(.25,16,.8)
[1] 12
```

This is the smallest number of successes such that the probability of at most this many successes is greater than or equal to .25.

Check this:

```
> pbinom(11, 16, .8)
> pbinom(12, 16, .8)
```

Example (cont.) Geometric

Find the probability of getting the first head on the fourth toss. This is the geometric distribution. The arguments to `geom` are `geom(failures, p)`.

```
> dgeom(3,.8)
```

The probability that the first head occurs on one of the first four tosses (that is, on the first, second, third or fourth toss) is

```
> pgeom(3,.8)
```

Example Poisson

Suppose a certain region of California experiences about 5 earthquakes a year. Assume occurrences follow a Poisson distribution. What is the probability of 3 earthquakes in a given year?

Here $\lambda = 5$

```
> dpois(3,5)
```

Check the answer:

```
> 5^3*exp(-5)/(3*2)
```

Random numbers

To generate random numbers from a particular distribution, preface the root name with an `r`.

For example, we continue our previous example of a biased coin with $p = .8$ of coming up heads. Toss this coin 25 times. The command `rbinom(1,25,.8)` will return a random number of successes.

```
> rbinom(1,25,.8)
```

Now, lets run this experiment 10 times (that is, we do 10 sets of tossing a coin 25 times) and record the number of successes.

```
> set.seed(0)
```

This sets the seed for the random number generator so that we all get the same results.

```
> heads <- rbinom(10, 25, .8)
> heads
```

```
[1] 17 19 21 18 20 18 22 18 22 17  #output will vary
```

In the first experiment of tossing the coin 25 times, 17 heads occurred. In the second experiment of tossing the coin 25 times, 19 heads occurred, etc.

```
> table(heads)
> barplot(table(heads))
```

Repeat the above, except now run the experiment 100 times.

```
> heads2 <- rbinom(100, 25, .8)
```