Introduction to R

Computing has dramatically changed the practice of statistics in the past twenty years, and these changes give us the opportunity to teach statistics in a more exciting and compelling manner. In addition, new computing tools offer valuable aids to help students understand statistical concepts and promote statistical thinking.

The R language is open source software available for use on Windows, Mac, and unix operating systems and supported by a world-wide community of statisticians. It is available at http://cran.us.r-project.org/.

R is an interactive and interpreted language. By interpreted, we mean that we can give an instruction and immediately have it evaluated. Then, we can give another command. With its simple command line interface students quickly learn how to employ basic statistical functions to work creatively with data.

Students Background in Computing

We will review the data collected by Moore and Legler on the use of computers in statistics courses in liberal arts colleges, *Survey on Statistics within the Liberal Arts College*, and data we have collected on our stat majors familiarity with computing at UC Berkeley and Davis.

We will also discuss the Summer Program in Statistics for Undergraduates, Data Visualization and its Role in Statistics, which took place June 19-24 in UCLA. More information on the workshop can be found at http://summer.stat.ucla.edu/.

Starting R

We first start R by invoking the command *R*. An R session will begin and provide you with the following information about the particular version of R being run, licensing information, and how to get help.

It also tells you to type 'demo()' for some demos, 'help()' for on-line help, or 'help.start()' for a HTML browser interface to help. Type 'q()' to quit R.

Also note, that the opening message tells how to exit an R session: invoke the command `q()`

```
> q()
Save workspace image? [y/n/c]:
```

Answering `y` will save this session. The saved session will be reloaded the next time you invoke R from the current directory. Note that the line that says

```
[Previously saved workspace restored]
```

means that R is loading up the data that we had saved from the last session.

Using R as a Calculator

We can use R as a heavyweight calculator

```
> 1+2
[1] 3
```

The operators are + for addition, - for subtraction * for multiplication, / for division and ^ for exponentiation. The order of operation is what you expect, with parenthesis overriding the defaults. We can use built-in values, such as **pi**.

```
> 1+pi
[1] 4.141593
```

There are also built in functions such as `log()` `exp()` `sqrt()` `cos()` `sin()` `tan()` `acos()` `asin()` `atan()`

Variables in R hold sets of numbers (i.e. they are vectors). There are two basic ways to assign a value to a variable in R.

```
> x = sqrt(9)
> x
[1] 3
> x <- 10+2
> x
[1] 12
```

Note that R uses "copying" semantics. When we assign the value of x to y, y gets the value of x. It is not "linked" to x so that when x is changed, y would see that change. Instead, we copy the value of x in the assignment and the two variables are unrelated after that.

```
> x
[1] 12
> y = x
> x = 10
> x
[1] 10
> y
[1] 12
```

Vectors

In R, the basic or primitive types of objects are vectors. These are simply collections of values grouped together into a single container. The variables `x` and `y` above are both vectors of one element. That is, in R there are no scalars. Instead, individual values are actually vectors of length 1 so they are special cases of general vectors with multiple elements. This makes lots of computations convenient. Below we create 10 random numbers from the normal distribution and assign them to the variable `x`.

```
> x = rnorm(10)
> x
[1] -0.52298969  0.05799518 -0.20754587  1.51625814  1.45267502 -0.79870467
[7]  1.30291855  0.11937840  1.85310113  0.85290233
```

The basic operations in R are vector operations so when we say `x + 2`, the value 2 is added to each element of `x`.

```
> x + 2
 [1] 1.477010 2.057995 1.792454 3.516258 3.452675 1.201295 3.302919
 [8] 2.119378 3.853101 2.852902
```

The catenate function, `c` can be used to create vectors. For example, to create a vector z of 4 numbers, -1, 0, 20, 3, we make the following assignment:

```
> z = c(-1, 0, 20, 3)
> z
[1] -1  0 20  3
```

Reading Data into R

Several functions are available to read data into R. One that is easy to use is `read.table`. and `read.csv`. The `read.csv` function expects the data to be comma separated and the first line to contain the names of the variables. The data to be organized with one row for each observation. Below we read into R weather data from Fargo, ND for 2003. Notice that the function can read a file from a url.

```
> w = read.csv("http://www.stat.berkeley.edu/users/nolan/BTF05/Fargo.csv")
> names(w)
[1] "Month" "Day"   "MAX"   "MIN"   "HDD"   "SPD"
> dim(w)
[1] 365   6
```

EXERCISE: Convert the daily maximum temperature (in MAX) from Fahrenheit to Celsius. Note to refer to the variable `MAX` we use `w$MAX`.

Notice we could have used the `read.table` function provided we passed it two additional parameters: the header parameter to say that the first line in the file contains a header with variable names, and the sep parameter tells the function that the values are separated by commas -- `read.csv("http://www.stat.berkeley.edu/users/nolan/BTH05/Fargo.csv", header = TRUE, sep = ",")`

To find out the parameters and their default values for a function, you can call the `args()` function. If that is not sufficient, help is available from the help function.

```
>
> args(read.table)
function (file, header = FALSE, sep = "", quote = "\"'", dec = ".",
    row.names, col.names, as.is = FALSE, na.strings = "NA", colClasses = NA,
    nrows = -1, skip = 0, check.names = TRUE, fill = !blank.lines.skip,
    strip.white = FALSE, blank.lines.skip = TRUE, comment.char = "#")
NULL
> help(read.table)
```

The R object `w` is a data frame, that is it is a rectangular array where each column is a vector. To refer to a variable in the data frame, we use its name as follows `w$MAX`. The basic types are integer, numeric, logical and character vectors. And a very important characteristic of these vector types is that they can only store values of the same type. In other words, a vector has homogeneous data types. We cannot use a vector to store both an integer and a string in their basic forms.

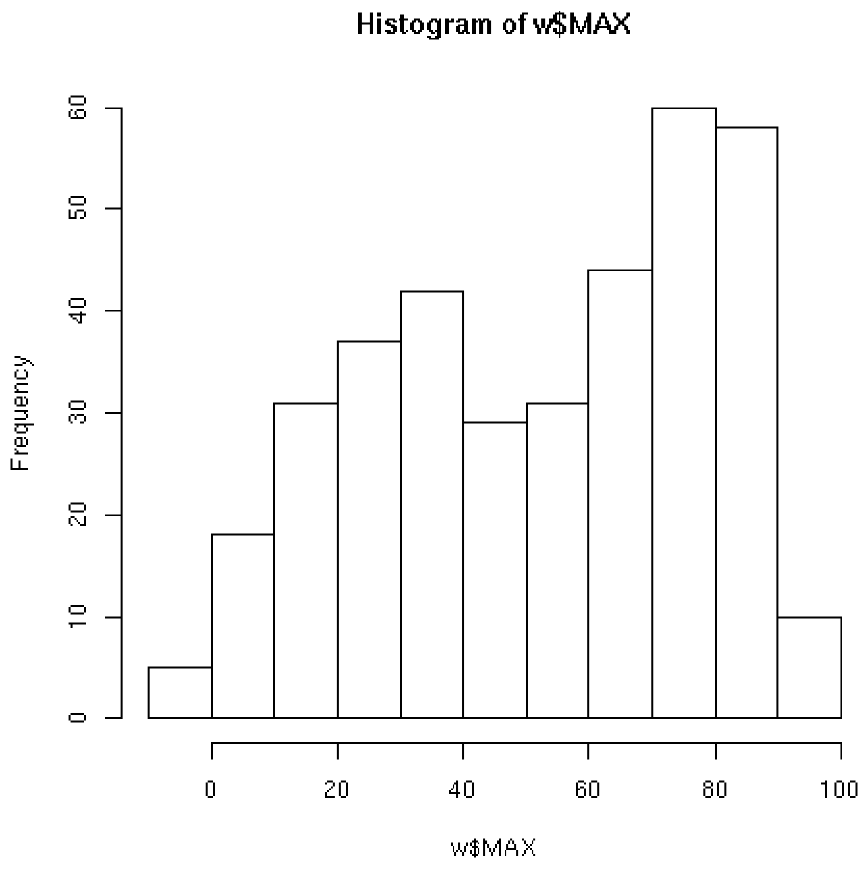
As we just said, there are 4 basic types of vectors: integer, character, numeric and logical. Integer vectors store integer values, numeric vectors store real numbers, logical vectors store values that are either TRUE or

FALSE and character vectors store strings.

Simple Plotting

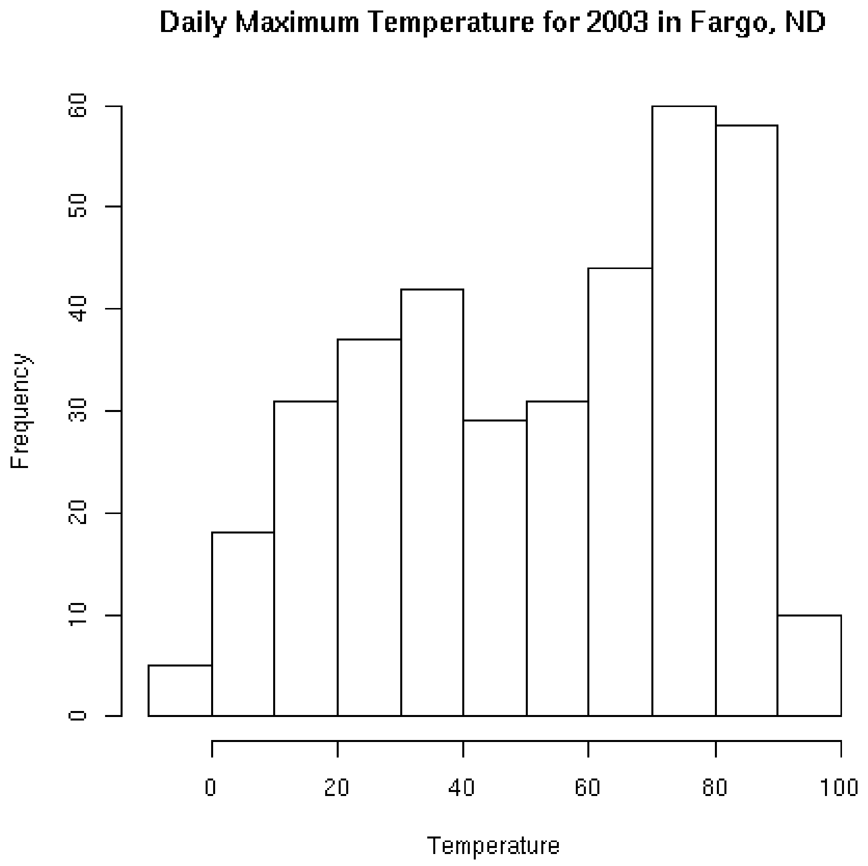
It is very simple to make attractive graphics in R. For example to examine the distribution of maximum daily temperature in Fargo in 2003, we simply call the `hist` command.

```
> hist(w$MAX)
```



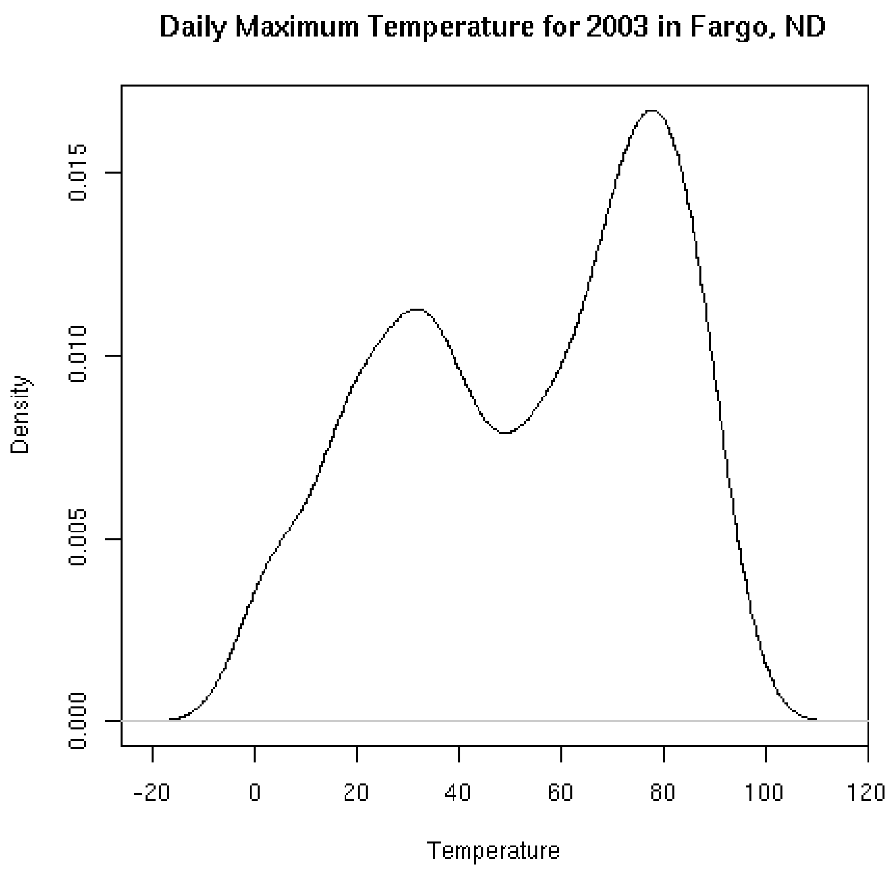
We can easily add more informative labels to the axes and a title to the histogram by adding a few parameters to the hist command.

```
> hist(w$MAX,xlab="Temperature",
   main="Daily Maximum Temperature for 2003 in Fargo, ND",freq=TRUE)
```



But why make histograms when we can make smooth densities?

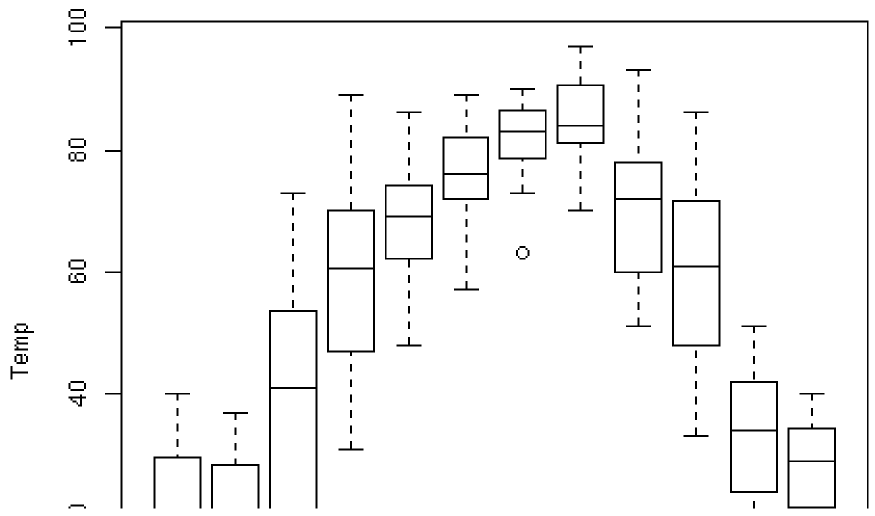
```
> maxD = density(w$MAX,adjust=0.8)
> plot(maxD,xlab="Temperature",
       main="Daily Maximum Temperature for 2003 in Fargo, ND")
```

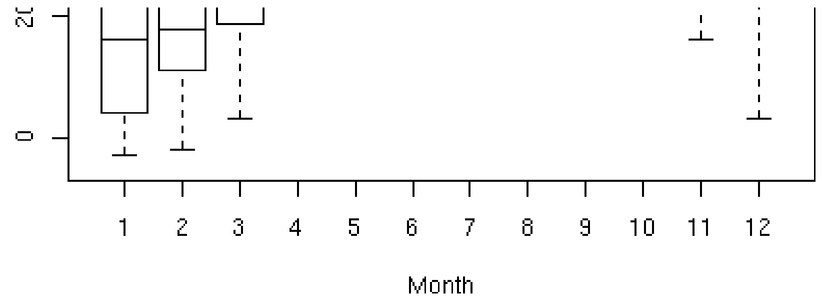


EXERCISE Make a series of boxplots for Maximum temperature by month.

```
> boxplot(w$MAX ~ w$Month, ylab = "Temp", xlab = "Month")
```

Notice that the boxplot function uses the formula `w$MAX ~ w$Month` to specify that the response variable, daily maximum is to be plotted against the different values of the month variable. This notation is very convenient for specifying the relationship between variables and carries over to model fitting.

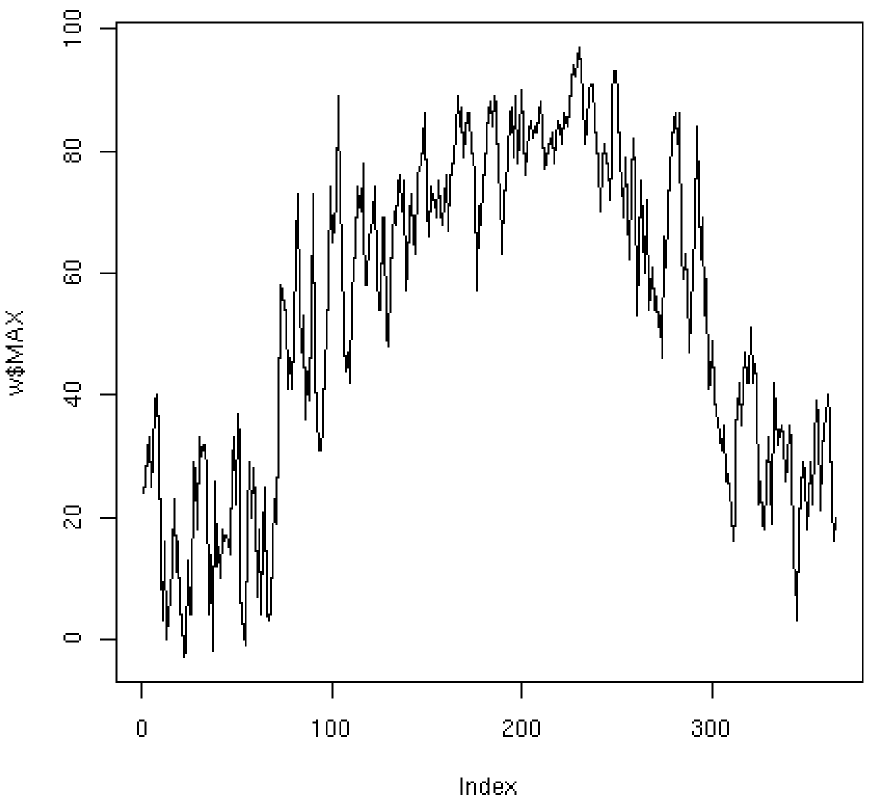




EXERCISE Make a time-series plot of the daily maximum and minimum temperatures.

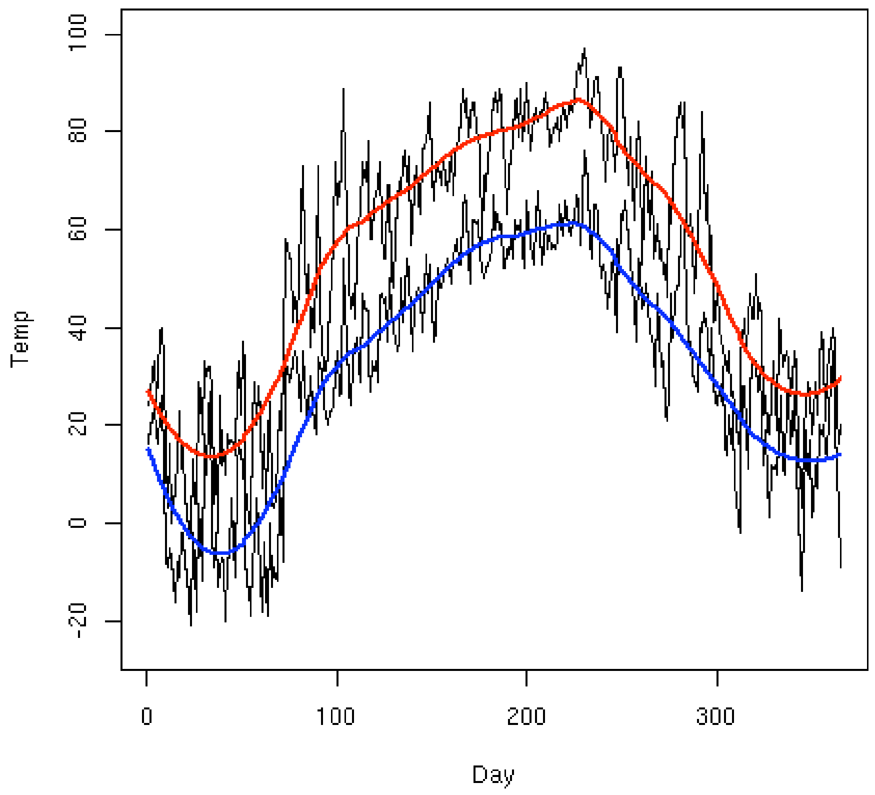
```
> plot(w$MIN,type="l")
```

Notice here that we first make a plot with the minimum temperature. Then we add to the plot the time series of the maximum temperature. Try running the code without the ylim parameter and the type parameter to see what happens to the plots.

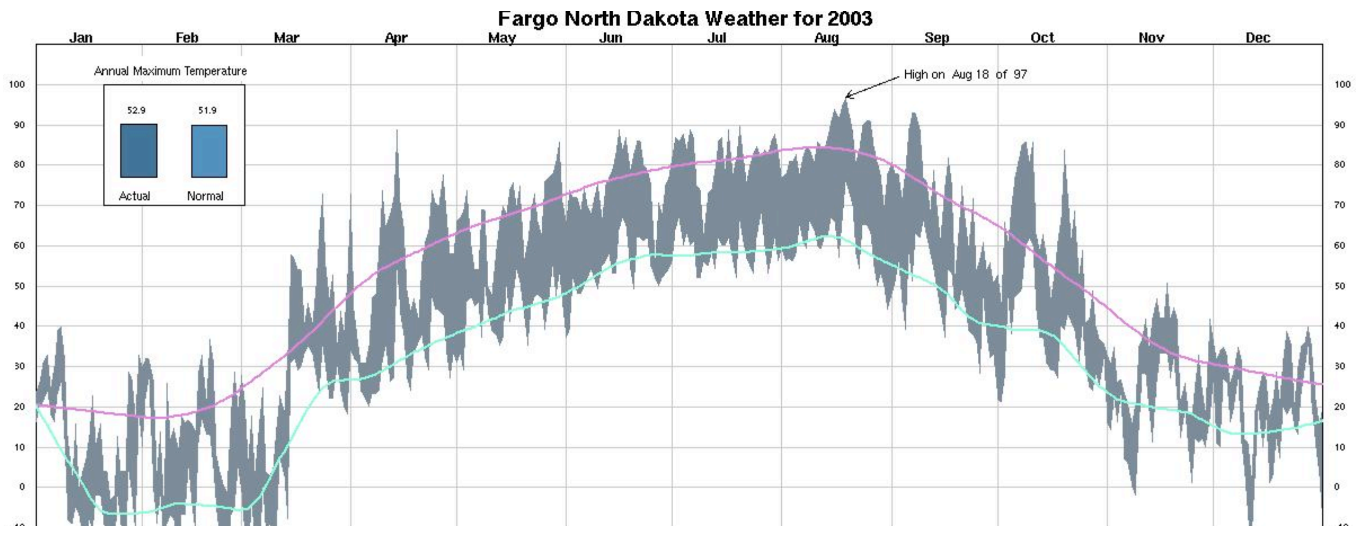


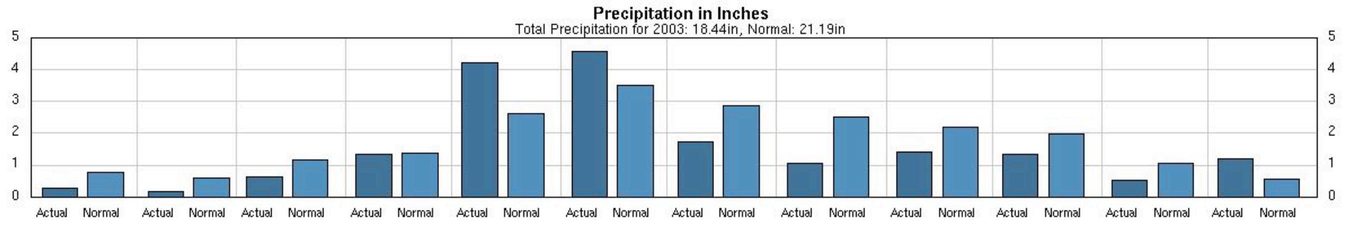
Once you learn how to use plot, you can build up more complex plots, one step at a time. The plot below includes both the minimum and maximum temperatures as well as smooth curves of these temperatures. Recall that the catenate function creates a vector from the values that it is passed. In this example the limits of the y axes are set to be -25 and 100 by passing the ylim parameter the vector of two extreme values.

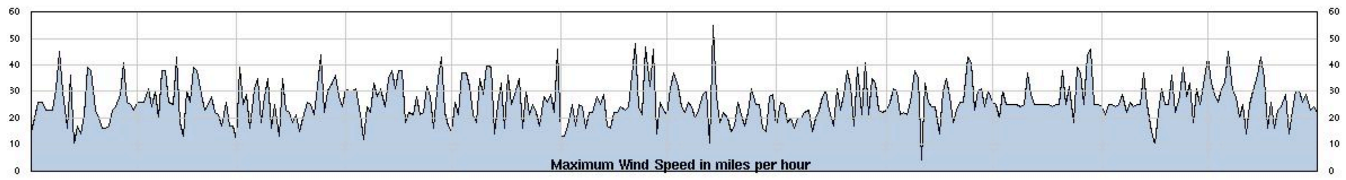
```
> Days = 1:365
> maxSmooth = loess(w$MAX~Days,span=0.3)
> maxPredict = predict(maxSmooth)
> minSmooth = loess(w$MIN~Days,span=0.3)
> minPredict = predict(minSmooth)

> plot(w$MIN~Days,type="l",ylim=c(-25,100),ylab="Temp")
> points(w$MAX,type="l")
> points(maxPredict,type="l",col="red",lwd=2)
> points(minPredict,type="l",col="blue",lwd=2)
```



Finally, here is a plot that students we able to make copying one that appears in Tufte's book for New York City.







Exercise: Exploring Freeway Traffic in LA

Questions courtesy of John Rice

The file traffic.txt contains data collected by loop detectors at a particular location of eastbound Interstate 80 in Sacramento, California, from March 14-20, 2003. For each of three lanes, the flow (number of cars) and the occupancy (the percentage of time there was a car over the loop) were recorded in successive five minute intervals. There were 1740 such five minute intervals. Lane 1 is the leftmost lane or fast lane, lane 2 is in the center, and lane 3 is the rightmost or slow lane.

To read the data into R, start with the following command:

```
> tt = read.table(file="http://www.stat.berkeley.edu/users/nolan/BTH05/traffic.txt",
         header=TRUE)
> names(tt)
[1] "Occ1"  "Flow1" "Occ2"  "Flow2" "Occ3"  "Flow3" "hour"  "day"
> dim(tt)
[1] 1740 8
```

Time Series

For each lane, plot flow and occupancy versus time. Explain the patterns you see.

Flow

Compare the flows in the three lanes by making parallel boxplots. Which lane typically serves the most traffic?

Examine the relationships of the flows in the three lanes by making scatter plots. Can you explain the patterns you see? Are statements of the form, ``The flow in lane 2 is typically about 50% higher than in lane 3," accurate descriptions of the relationships? If not, how would you verbally summarize the nature of the relationships?

Suppose the flow in lane 3 was missing because the detector was down. Examine how it could be predicted from the flow in lane 1 via scatter plot smoothing (see loess and predict or scatter.smooth).

Occupancy

Occupancy can be viewed as a measure of congestion. Find the mean and median occupancy in each of the three lanes. Do you think that the distributions of occupancy are symmetric or skewed? Why?

Are there any unusual features, and if so, how might they be explained?

Make plots to support or refute the statement, ``When one lane is congested, the others are, too."

Flow and Occupancy

Flow can be regarded as a measure of the throughput of the system. How does this throughput depend on congestion? Consider the following conjecture: ``When there are very few cars on the road, flow is small and so is congestion. Adding a few more cars may increase congestion but not enough so that velocity is decreased, so flow will also increase. Now beyond some point, increasing occupancy (congestion) will decrease velocity,but since there will then be more cars in total, flow will still continue to increase." Does this seem plausible to you?

Plot flow versus occupancy for each of the three lanes. Does this conjecture appear to be true? Can you explain what you see? Can you see "critical points" at which flow breaks down in the plots? Is the relationship of flow to occupancy the same in all lanes?

Suppose that all vehicles had a common length, L (this is not quite true, of course). How would velocity be related to flow and occupancy? Deduce how velocity depends upon occupancy in the plots of flow versus occupancy.

Simulation Studies in R

The `sample` takes samples from a provided population. For example, the variable pop below contains 17 values, and the first call to sample with arguments pop and 3 give us a sample of 3 without replacement from pop, the second gives us a sample of 20 with replacement, and the last call uses the default sample size which is the length of pop.

```
> pop = c(1, 20, 31, 8, 9, 6, 12, 12, 12, 12, 1, 1, 1, 1, 1, 1, 1)
> sample(pop,3)
[1]  9 12 12

> sample(pop, 20, replace=TRUE)
 [1] 12 20 12  1  1  6  1  1  1  1  1 20  1  1  9  1  8  1 31  8

> sample(pop)
 [1]  1 20  6 12  1 31  1 12  1 12  1 12  1  1  1  9  8
```

We can repeat this sampling many times to examine the sampling distribution of a statistic. The sapply function allows us to that.

```
> sampMean = sapply(1:1000,sample,x = pop, size=7)
```

Notice that we pass the parameters to the sample function as arguments to the sapply function.

Central Limit Theorem Example: Consider the behavior of the sum from a sample of size 7 from the uniform distribution. We can find the sum of a sample of 7 uniformly generated observations with the simple call to runif and sum.

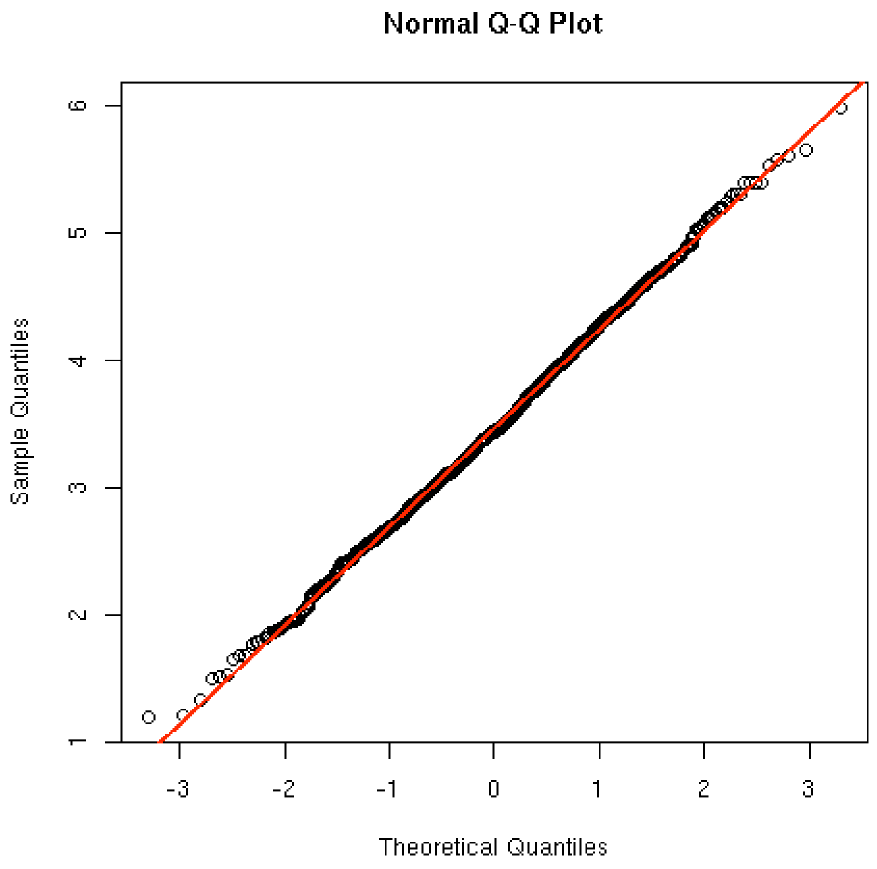
```
> sum(runif(7))
[1] 3.012257
```

But to find an approximate distribution of this sum, we need to repeat this process many times and examine the empirical distribution. We do this below, using the matrix function to create a matrix of 1000 by 7 which we then sum across the rows to get 1000 values, each one the sum of 7 uniformly generated observations.

```
> n = 7
> R = 1000
> randu = matrix(runif(n*R), nrow = R)
> stat1000 = apply(randu, 1, sum)
```

A plot of the density of the 1000 values and a normal quantile plot indicate that the sum of 7 uniforms is close to normal.

```
> plot(density(stat1000))
> qqnorm(stat1000)
> abline(a=mean(stat1000),b=sqrt(var(stat1000)),col="red",lwd=2)
```



If this code is too complex, we can write a simple function for our students to call. The function wraps the code up into a single function call, and allows the students to easily adjust the sample size, number of replications, and the statistic. These are the parameters n, R, and stats to the function myRan.

```
> myRan = function(n = 7, R = 1000, stats = "sum") {
+ randu = matrix(runif(n*R), nrow = R)
+ apply(randu, 1, stats)}

> xx = myRan()
> xy = myRan(n = 3, stats="median")
```

R offers many other random number generators, including `rnorm, rexp, rgamma, rweibull, rchisq, rf, rt, rlnorm` for the normal, exponential, gamma, weibull, chi-square, F, t, and log normal distributions respectively. In addition, R offers discrete random number generators including `rbinom, rpois, rgeom, rnbinom, rhyper,` for the binomial, Poisson, geometric, negative binomial and hypergeometric distributions.

Aside from offering the random number generators, it is possible to find the density value, percentile, and quantiles for these distributions. The function names simply replace the ``r'' with the appropriate letter, d for density, p for percentile, and q for quantile.

```
dnorm(x, mean=0, sd=1, log = FALSE)
pnorm(q, mean=0, sd=1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean=0, sd=1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean=0, sd=1)
```

Saving Variables from an R Session

When we end an R session using the q() function, we will normally be asked whether we want to save the session or not. If we don't want to store all the variables, but only specific ones, we can explicitly `save` one or more objects to a file. This is convenient when we create a big dataset and then want to ensure that it gets saved before we do anything else. Or if we want to make an object available to another R session, e.g. to somebody we are working with, without terminating ours, we can simply write the object to disk and then send it that person in an entirely portable format.

To find out which variables we may want to save, we use the function `objects`, which gives us the names of the variables we have in our workspace.

```
> objects()
[1] "x"  "y"  "w"
```

We can remove values from the workspace using `remove` by passing the name to the function of the variable we want to remove.

```
> remove(x)
```

Further Work

With R, students also learn a valuable skill that they can carry with them in more advanced courses because R offers advanced plotting routines, up-to-date statistical methodologies, and a rich programming environment. In addition, with R, instructors can build graphical user interfaces to perform customized analyses focussed on specific skills, plug R into spreadsheets to take advantage of their simple visual metaphors for computation, and interface R with databases and other languages such as Perl and C++. In this talk, we provide a brief introduction to the R language, and we cover examples from our teaching of how to use R's command line interface as well as more sophisticated interfaces for these purposes.

We will demo some of these approaches in the talk.

References

There are many excellent reference manuals for R available on the R website http://cran.us.r-project.org/.
For beginners you might find the following references useful.

- R for Beginners, Emmanuel Paradis
- Using R for Data Analysis and Graphics - Introduction, Examples, and Commentary, John Maindonald
- R Reference Card, Jonathan Baron