```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information.

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.

[Previously saved workspace restored]

> ls()
[1] "x" "y" "z"

> max(x)
[1] 3
```

All the examples so far (and many of the examples to follow) are interactive, but for serious work, it's better to work with a command file. Put your commands in a file and execute them all at once. Suppose your commands are in a file called `commands.R`. At the S prompt, you'd execute them with `source("commands.R")`. From the unix prompt, you'd do it like this. The `--vanilla` option invokes a "plain vanilla" mode of operation suitable for this situation.

```
credit.erin > R --vanilla < commands.R > homework.out
```

For really big simulations, you may want to run the job in the background at a lower priority. The `&` suffix means run it in the background. `nohup` means don't hang up on me when I log out. `nice` means be nice to other users, and run it at a lower priority.

```
credit.erin > nohup nice R --vanilla < bvnorm.R > bvnorm.out &
```

6.3 S as a Stats Package

Here, we illustrate traditional multiple regression with S, testing the parallel slopes assumption for the metric cars data. Compare `mcars.sas` and `mcars.lst`. There are lots of comment statements that help explain what is going on. More detail will be given in lecture. In addition, the course home page has a link to a nice 100-page manual. If you plan to use R seriously, you should download this manual and read it. But if you come to lecture, you probably don't need to look at it for the purposes of this class.

Here is the "program" named `lesson2.R`.

```
######################################################################
# lesson2.R: execute with    R --vanilla < lesson2.R > lesson2.out #
######################################################################

datalist <-  scan("mcars.dat",list(id=0,country=0,kpl=0,weight=0,length=0))
# datalist is a linked list.
datalist
# There are other ways to read raw data. See help(read.table).
weight <- datalist$weight ; length <- datalist$length ; kpl <- datalist$kpl
country <- datalist$country
cor(cbind(weight,length,kpl))
# The table command gives a bare-bones frequency distribution
table(country)
# That was a matrix. The numbers 1 2 3 are labels.
# You can save it, and you can get at its contents
countrytable <- table(country)
countrytable[2]
# There is an "if" function that you could use to make dummy variables,
# but it's easier to use factor.
countryfac <- factor(country,levels=c(1,2,3),
                     label=c("US","Japanese","European"))
# This makes a FACTOR corresponding to country, like declaring it
# to be categorical. How are dummy variables being set up?
contrasts(countryfac)
# The first level specified is the reference category. You can get a
# different reference category by specifying the levels in a different order.
cntryfac <- factor(country,levels=c(2,1,3),
                     label=c("Japanese","US","European"))
contrasts(cntryfac)
# Test interaction. For comparison, with SAS we got F = 11.5127, p < .0001
# First fit (and save!) the reduced model. lm stands for linear model.
redmod <- lm(kpl ~ weight+cntryfac)
# The object redmod is a linked list, including lots of stuff like all the
# residuals. You don't want to look at the whole thing, at least not now.
summary(redmod)

# Full model is same stuff plus interaction. You COULD specify the whole thing.
fullmod <- update(redmod,. ~ . + weight*cntryfac)
anova(redmod,fullmod)
# The ANOVA summary table is a matrix. You can get at its (i,j)th element.
aovtab <- anova(redmod,fullmod)
aovtab[2,5] # The F statistic
```

```
aovtab[2,6] < .05       #    p < .05 -- True or false?
1>6 # Another example of an expression taking the logical value true or false.
```

Here is the output file `lesson2.out`. Note that it shows the commands. This would not happen if you used `source("lesson2.R")` from within R. I have added some blank lines to the output file to make it more readable.

```
> ######################################################################
> # lesson2.R: execute with    R --vanilla < lesson2.R > lesson2.out #
> ######################################################################
>
> datalist <-  scan("mcars.dat",list(id=0,country=0,kpl=0,weight=0,length=0))
Read 100 records
> # datalist is a linked list.
> datalist
$id
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100

$country
  [1] 1 2 1 1 1 1 3 1 3 1 2 1 1 3 2 1 1 1 3 2 1 1 1 3 2 1 1 1 1 1 2 1 2 1 1 1 3
 [38] 3 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 3 1 1 1 2 1 1 1 1 2 2 1 1 1 2 1 1
 [75] 1 2 2 3 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 3 3 3 1 1 1

$kpl
  [1]  5.04 10.08  9.24  7.98  7.98  7.98  9.66  7.56  5.88 10.92 12.60  8.40
 [13]  8.82 10.92  7.56 12.18  5.04  5.88  7.14 13.02  5.88 10.92  6.72 10.50
 [25]  8.82  5.88  6.72 11.76  9.24  7.56  7.56 11.76 10.50  5.88  9.24  7.98
 [37]  7.14 17.22  6.72  7.98  7.14  6.30  5.88  8.82  9.24  9.24  5.88  8.40
 [49] 10.50  9.24  7.56  7.56 12.60 12.60  7.98  7.56  8.40  9.66  7.56  6.30
 [61]  5.88  7.56 10.08  5.04  8.82 11.76 14.70 10.08  9.24 10.92 10.50  7.56
 [73]  8.82  7.56  7.14  7.56 10.08  8.82  5.88  8.82  8.82 10.08 17.22  6.72
 [85]  9.24  5.88  7.56 11.76  7.98  8.82  5.88  5.88  7.14  5.04 17.22 17.22
 [97]  7.14 10.50  6.72  7.56

$weight
  [1] 2178.0 1026.0 1188.0 1444.5 1485.0 1485.0  972.0 1665.0 1539.0 1003.5
 [11]  891.0 1273.5 1930.5  823.5 1084.5  949.5 2178.0 1755.0 1426.5  990.0
 [21] 1827.0 1134.0 1813.5 1192.5 1237.5 1858.5 1813.5 1062.0 1431.0 1651.5
 [31] 1201.5 1062.0 1008.0 1858.5 1318.5 1440.0 1273.5  918.0 1813.5 1530.0
 [41] 1683.0 1836.0 1723.5 1827.0 1449.0 1318.5 1858.5 1273.5  868.5 1318.5
 [51] 1665.0 1620.0  954.0  954.0 1516.5 1665.0 1462.5  972.0 1665.0 1674.0
 [61] 1755.0 1201.5 1237.5 2178.0 1930.5 1062.0  922.5 1026.0 1449.0 1134.0
 [71]  990.0 1084.5 1930.5 1516.5 1507.5 1084.5 1026.0  958.5 1858.5 1930.5
 [81] 1192.5 1237.5  918.0 1813.5 1449.0 1755.0 1561.5 1062.0 1489.5 1192.5
 [91] 1827.0 1755.0 1683.0 2178.0  918.0  918.0 1426.5  990.0 1660.5 1498.5

$length
  [1] 591.82 431.80 426.72 510.54 502.92 502.92 436.88 543.56 487.68 431.80
 [11] 391.16 495.30 518.16 360.68 441.96 414.02 591.82 518.16 490.22 419.10
 [21] 561.34 462.28 523.24 449.58 467.36 551.18 523.24 431.80 490.22 553.72
```

```
 [31] 444.50 431.80 436.88 551.18 472.44 505.46 480.06 393.70 523.24 508.00
 [41] 558.80 563.88 510.54 558.80 508.00 472.44 551.18 495.30 393.70 472.44
 [51] 543.56 523.24 414.02 414.02 508.00 543.56 497.84 436.88 543.56 538.48
 [61] 518.16 444.50 454.66 591.82 518.16 431.80 416.56 431.80 508.00 462.28
 [71] 419.10 441.96 518.16 502.92 439.42 441.96 431.80 408.94 551.18 518.16
 [81] 454.66 454.66 393.70 523.24 508.00 518.16 502.92 431.80 502.92 454.66
 [91] 561.34 518.16 558.80 591.82 393.70 393.70 490.22 419.10 538.48 510.54

> # There are other ways to read raw data. See help(read.table).

> weight <- datalist$weight ; length <- datalist$length ; kpl <- datalist$kpl
> country <- datalist$country
> cor(cbind(weight,length,kpl))
           weight     length        kpl
weight  1.0000000  0.9462018 -0.7704194
length  0.9462018  1.0000000 -0.7899859
kpl    -0.7704194 -0.7899859  1.0000000

> # The table command gives a bare-bones frequency distribution
> table(country)
country
 1  2  3
73 13 14

> # That was a matrix. The numbers 1 2 3 are labels.
> # You can save it, and you can get at its contents
> countrytable <- table(country)
> countrytable[2]
 2
13

> # There is an "if" function that you could use to make dummy variables,
> # but it's easier to use factor.
> countryfac <- factor(country,levels=c(1,2,3),
+                      label=c("US","Japanese","European"))
> # This makes a FACTOR corresponding to country, like declaring it
> # to be categorical. How are dummy variables being set up?

> contrasts(countryfac)
         Japanese European
US              0        0
Japanese        1        0
European        0        1

> # The first level specified is the reference category. You can get a
> # different reference category by specifying the levels in a different order.
> cntryfac <- factor(country,levels=c(2,1,3),
+                      label=c("Japanese","US","European"))

> contrasts(cntryfac)
         US European
Japanese  0        0
US        1        0
European  0        1
```

```
> # Test interaction. For comparison, with SAS we got F = 11.5127, p < .0001

> # First fit (and save!) the reduced model. lm stands for linear model.
> redmod <- lm(kpl ~ weight+cntryfac)

> # The object redmod is a linked list, including lots of stuff like all the
> # residuals. You don't want to look at the whole thing, at least not now.

> summary(redmod)

Call:
lm(formula = kpl ~ weight + cntryfac)

Residuals:
    Min      1Q  Median      3Q     Max
-3.0759 -0.9810 -0.1919  0.4725  5.0795

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)     16.2263357  0.7631228  21.263   <2e-16 ***
weight          -0.0060407  0.0005708 -10.583   <2e-16 ***
cntryfacUS       1.2361472  0.5741299   2.153   0.0338 *
cntryfacEuropean  1.4595914  0.6456563   2.261   0.0260 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.676 on 96 degrees of freedom
Multiple R-Squared: 0.618,      Adjusted R-squared: 0.606
F-statistic: 51.76 on 3 and 96 DF,  p-value:     0

>
> # Full model is same stuff plus interaction. You COULD specify the whole thing.

> fullmod <- update(redmod,. ~ . + weight*cntryfac)

> anova(redmod,fullmod)

Analysis of Variance Table

Model 1: kpl ~ weight + cntryfac
Model 2: kpl ~ weight + cntryfac + weight:cntryfac
  Res.Df     RSS Df Sum of Sq      F    Pr(>F)
1     96 269.678
2     94 216.617  2    53.061 11.513 3.372e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> # The ANOVA summary table is a matrix. You can get at its (i,j)th element.
> aovtab <- anova(redmod,fullmod)
> aovtab[2,5] # The F statistic
[1] 11.51273

> aovtab[2,6] < .05       #    p < .05 -- True or false?
[1] TRUE

> 1>6 # Another example of an expression taking the logical value true or false.
[1] FALSE
```

6.4 Random Numbers and Simulation

S is a superb environment for simulation and customized computer-intensive statistical methods. That's really why it is being discussed. Simulation is an extremely general and powerful method for calculating probabilities that are difficult to figure out by other means. Well, technically it's a way of *estimating* those probabilities, based a sample of random numbers. Before proceeding, we need a couple of definitions.

We will use the term **statistical experiment** to refer to any procedure whose outcome is not known in advance with certainty. The most standard, and the most boring example of a statistical experiment is to toss a coin and observe whether it comes up heads or tails. We *model* statistical experiments by pretending that they obey the laws of probability.

When we carry out a statistical experiment, the things that can happen (the things we pay attention to) are called **outcomes**. Sets of outcomes are called **events**. For example, if you roll a die, the outcomes are the numbers 1 through 6, and "even" is an event consisting of the outcomes $\{2, 4, 6\}$.

The main principle we will use is called the **Law of Large Numbers**. There are quite a few versions of this law. Here's a verbal statement of the one we will use. *If a statistical experiment is carried out independently a very large number of times (trials) under identical conditions, the proportion of times an event occurs approaches the probability of the event, as the number of trials increases.* In elementary texts, this is sometimes used as the definition of probability. But in more sophisticated treatments, it's a theorem.

For example, suppose you are planning to test differences between means for an experimental versus a control group, and you have strong reason to believe that your data will have a chi-square distribution within groups. You are going to log-transform the data to take care of the positive skewnes of the chi-square, and then use a common t-test.

Suppose data in the experimental group is chi-square with one degree of freedom (so the population mean is one and the variance is two), and the data in the control group is chi-square with two degree of freedom (so the population mean is two and the variance is four). What is the power of the t-test on the transformed data with $n = 20$ in each group?

Nobody can figure this out mathematically, but it's pretty easy with simulation. Here's how to do it.

1. Using the random number generator in some software package, generate 20 independent chi-square values with one degree of freedom, and 20 independent chi-square values with two degrees of freedom.

2. Log transform all the values.

3. Compute the t-test.

4. Check to see if $p < 0.05$.

Do this a large number of times. The proportion of times $p < 0.05$ is the power — or more precisely, a Monte Carlo estimate of the power.

The number of times a statistical experiment is repeated is called the **Monte Carlo sample size**. How big should the Monte Carlo sample size be? It depends on how much precision you need. We will produce confidence intervals for all our Monte Carlo estimates, to get a handle on the probable margin of error of the statements we make. Sometimes, Monte Carlo sample size can be chosen by a power analysis. More details will be given later.

```
> rnorm(20) # 20 standard normals
 [1]  0.24570675 -0.38857202  0.47642336  0.75657595  0.71355871 -0.74630629
 [7] -0.02485569  1.93346357  0.15663167  1.16734485  0.57486449  1.32309413
[13]  0.63712982  2.00473940  0.04221730  0.70896768  0.42128470 -0.12115292
[19]  1.42043470 -1.04957255

> set.seed(12345) # Be able to reproduce the stream of pseudo-random numbers.
> rnorm(20)
 [1]  0.77795979 -0.89072813  0.05552657  0.67813726  0.80453336 -0.35613672
 [7] -1.24182991 -1.05995791 -2.67914037 -0.01247257 -1.22422266  0.88672878
[13] -1.32824804 -2.73543539  0.40487757  0.41793236 -1.47520817  1.15351981
[19] -1.24888614  1.11605686
> rnorm(20)
 [1]  0.866507371  2.369884323  0.393094088 -0.970983967 -0.292948278
 [6]  0.867358962  0.495983546  0.331635970  0.702292771  2.514734599
[11]  0.522917841 -0.194668990 -0.089222053 -0.491125596 -0.452112445
[16] -0.515548826 -0.244409517 -0.008373764 -1.459415684 -1.433710170

> set.seed(12345)
> rnorm(20)
 [1]  0.77795979 -0.89072813  0.05552657  0.67813726  0.80453336 -0.35613672
 [7] -1.24182991 -1.05995791 -2.67914037 -0.01247257 -1.22422266  0.88672878
[13] -1.32824804 -2.73543539  0.40487757  0.41793236 -1.47520817  1.15351981
[19] -1.24888614  1.11605686
```