Statistics 5041
1. Introduction

Gary W. Oehlert
School of Statistics
313B Ford Hall
612-625-1557
gary@stat.umn.edu

Much of statistics is *Univariate*, meaning that we are dealing with a single random variable at a time.

- Compare mean salary for male and female CEOs.
- Predict soil moisture from temperature and precipitation.
- Analyze variation in product performance.

Even multiple regression is univariate, because there is a single response.
What about when we

- Compare mean salary for male and female CEOs.
- Compare mean company size for male and female CEOs.
- Compare mean growth rates for male and female CEOs.

People often study these variables one at a time.
Multivariate methods look at all the variables at once.
This can be more powerful, can reveal more subtle effects, can reveal relationships missed one at a time, and so on.
It is also more complex, and distributional properties for multivariate methods are often known only approximately.
Examples of multivariate data:
Measurements of weight at ages 1 through 15 for 10 chimpanzees (repeated measures).
Observations of orbital height, period, excentricity, and orientation for man-made satellites.
Age, blood pressure, hemoglobin, and time to completion for blood donors.
We usually express multivariate data in a *matrix* or *array* X, with a column for each variable, and a row for each case or unit. Individual elements are x_{ij}.

	Var 1	...	Var j	...	Var p
Unit 1	x_{11}	...	x_{1j}	...	x_{1p}
Unit 2	x_{21}	...	x_{2j}	...	x_{2p}
$\vdots$	$\vdots$		$\vdots$		$\vdots$
Unit i	x_{i1}	...	x_{ij}	...	x_{ip}
$\vdots$	$\vdots$		$\vdots$		$\vdots$
Unit n	x_{n1}	...	x_{nj}	...	x_{np}

n units in n rows, p variables in p columns.

$$\mathbf{X} = \begin{bmatrix} x_{11} & \dots & x_{1j} & \dots & x_{1p} \\ x_{21} & \dots & x_{2j} & \dots & x_{2p} \\ \vdots & & \vdots & & \vdots \\ x_{i1} & \dots & x_{ij} & \dots & x_{ip} \\ \vdots & & \vdots & & \vdots \\ x_{n1} & \dots & x_{nj} & \dots & x_{np} \end{bmatrix}$$

Examples of data that are *not* multivariate:
Weight measurements on 10 1 year old chimpanzees, 10 2 year old chimpanzess, . . ., 10 15 year old chimpanzees.
Sorted age, sorted blood pressure, and sorted hemoglobin from 50 subjects.
In both cases, there is no link between the variables and a single unit. That is, x_{11} and x_{12} are not linked to the same unit. We just have several univariate data sets.

Introduction to MacAnova

Think of MacAnova as a calculator. It has lots of ordinary functions (logarithms, squares, multiplication, etc), and it also has lots of statistical functions (means, histograms, analysis of variance, etc).
The basic structure of MacAnova is that you type a command at the prompt; then MacAnova does something for you, and possibly prints something out.

```
Cmd> 3+sqrt(5)
(1)       5.2361
```

Here we just computed $3 + \sqrt{5}$.
Read the "Introduction to MacAnova" or the "MacAnova User's Guide" for a complete introduction!
You put data into MacAnova variables and then use MacAnova functions to manipulate the variables.

```
Cmd> x <- 7  # note that nothing prints

Cmd> x*3
(1)           21
```

The combination `<-` is the assignment operator; think of it as a left pointing arrow. Here we assign to a variable. Later we can use the variable in arithmetic.
Anything typed after a # (a pound or sharp) is a comment and is not evaluated by MacAnova.
Variable names can be up to 12 letters and can include numbers and _; for example `age_8`.
MacAnova variables can keep numbers as a single number (a scalar), a list of numbers (a vector), a matrix of numbers, and so on. We will work a lot with vectors and matrices.
The `vector()` command collects its arguments into a vector.

```
Cmd> x <- vector(3.47,3.73,3.38,\
3.87,3.69,3.51,3.35,3.64);x
(1)   3.47   3.73   3.38   3.87   3.69
(6)   3.51   3.35   3.64
```

You can continue on the next line by ending a line with a backslash. You can add another command on a line by including a semicolon.

```
Cmd> x2 <- vector(x,x)
```

This makes a new vector with two copies of `x` in it.
A matrix is a two-dimensional data array constructed with the `matrix()` function.

```
Cmd> z <- matrix(x,4);z
(1,1)         3.47         3.69
(2,1)         3.73         3.51
(3,1)         3.38         3.35
(4,1)         3.87         3.64
```

The arguments to `matrix()` are the list of numbers for the contents and the number of rows.
Data are entered down columns.
Help is available for all functions and many other topics.

```
Cmd> help(matrix)
matrix(x,Rowdim) creates a matrix (two dimensional array) with Rowdim
rows containing the data in x.  x can be a vector, matrix, or higher
dimensional array, all of whose elements are used, with first subscript
changing fastest, second subscript, if any, changing next, and so on.
...
```

Want a complete list of help topics?

```
Cmd> help("*")
Help is available on the following topics:
abs             cumstudrng      invchi          nbits           strconcat
acos            customize       invdunnett      ncols           stringplot
addchars        data_files      invF            ncomps          structure
...
```

Do some stuff with our data:

```
Cmd> sum(x) # total
(1)        28.64

Cmd> sum(x)/8 #average
(1)         3.58

Cmd> describe(x,mean:T) # average too
(1)         3.58
```

```
Cmd> sort(x) # sort ascending
(1)   3.35   3.38   3.47   3.51   3.64
(6)   3.69   3.73   3.87

Cmd> sort(x,down:T) # sort descending
(1)   3.87   3.73   3.69   3.64   3.51
(6)   3.47   3.38   3.35

Cmd> reverse(sort(x)) # another way
(1)   3.87   3.73   3.69   3.64   3.51
(6)   3.47   3.38   3.35
```

Often several ways to do the same thing.
Many operations work on columns of matrices:

```
Cmd> sort(z)
(1,1)         3.38         3.35
(2,1)         3.47         3.51
(3,1)         3.73         3.64
(4,1)         3.87         3.69
```

Take a matrix apart:

```
Cmd> makecols(z,c1,c2)

Cmd> c1
(1)    3.47    3.73    3.38    3.87

Cmd> c2
(1)    3.69    3.51    3.35    3.64
```

Put it back together:

```
Cmd> hconcat(c1,c2)
(1,1)         3.47         3.69
(2,1)         3.73         3.51
(3,1)         3.38         3.35
(4,1)         3.87         3.64
```

horizontal **concat**enation

```
Cmd> a <- "hello";a  # character variable
(1) "hello"

Cmd> save("testsave")  # save workspace
Workspace saved on file testsave
```

```
Cmd> delete(a)  # delete a

Cmd> a  # now it's not defined
UNDEFINED

Cmd> restore("testsave") # restore workspace
Restoring workspace from file testsave
Workspace saved Tue Sep  3 13:40:03 2002

Cmd> a  # a is back
(1) "hello"
```

Subscripting

```
Cmd> z[1,] # first row
(1,1)         3.47         3.69

Cmd> z[,2] # second column
(1,1)         3.69
(2,1)         3.51
(3,1)         3.35
(4,1)         3.64

Cmd> z[1,2] # row 1, col 2
(1,1)         3.69

Cmd> z[-3,] # all but row 3
(1,1)         3.47         3.69
(2,1)         3.73         3.51
(3,1)         3.87         3.64

Cmd> z[vector(2,3),1] # col 1, row 2,3
(1,1)         3.73
(2,1)         3.38

Cmd> x;x>3.5 # logical variable
(1)    3.47    3.73    3.38    3.87    3.69
(6)    3.51    3.35    3.64
(1) F   T   F   T   T   T   F   T

Cmd> x[x>3.5] # x with x > 3.5
(1)    3.73    3.87    3.69    3.51    3.64
```

Most of the JW data sets on the CDROM are space separated text files giving a data table. We can read them in using `readdata()` by giving column names.

```
Cmd> readdata("/cdrom/P1-4.DAT",sales,\
profits,assets)
Read from file "/cdrom/P1-4.DAT"
Column 1 saved as REAL vector sales
Column 2 saved as REAL vector profits
Column 3 saved as REAL vector assets
```

Note, you can use `""` as the file name in windowed versions and get the usual file browsing dialog box.