```
-----------------------------------
  R INTRODUCTION PART I
-----------------------------------

# First, where do you get R?
# Go to http://www.R-project.org and download it from there, it's FREE!!

# Now, what is so great about R?
# --------------------------------
# There are several things:
#  -- it is free and runs on Windows, Macintosh, and UNIX machines
#  -- it is powerful command language with many statistical functions
#  -- it has excellent graphical capabilities
#  -- it comes with a nice exploratory data analysis package
#     that will be useful for our class
#  -- it allows you to create your own functions or even another R package
# ---------------------------------
# But ...
#  -- it is command oriented, so it is harder to learn than a menu-oriented
#     system
#  -- need to get familiar with the syntax
#  -- it has no commercial support (this might actually be a good thing)
# ---------------------------------
# The objective of this introductory lessons are to get you familiar with the
# use of R to do some data analysis and graphs.

# By the way, if you wish to type comments in R, you preface the comment
# with a "#" symbol (like I'm doing here).

# Note: R commands are case sensitive!

# In this session, we introduce some commands to define and manipulate vectors of data.

# Topics:
          combine command c()
          sequence command seq()
          replicate command rep()
          mathematical operations on vectors
          use of brackets
          logical values
          extracting elements

# Suppose we wish to enter the scores 12, 32, 54, 33, 21, 65 into R
# use c() command to create a vector; store result in variable x
x=c(12,32,54,33,21,65)

# this can be used to store vector of character data
y=c("Joe","Mary","Sue","Harry")

# we can use c() to add elements to vector x
x=c(x,55,32)

# By the way, the command above will only assign the vector to the variable x.
# If we wish to see x, we just type
x

# the seq() command is useful in creating regular sequences of numbers
# say I want to create a vector of the years from 1990 through 2003

years=seq(1990,2003)
years

# another way of doing this is by the colon (:)
years=1990:2003
years

# suppose we want a to contain values from 3 to 5 in steps of .5
a=seq(3,5,.5)
a

# the rep() command is useful for creating data that follow a regular pattern
rep(1,5)
rep(1:2,4)

# mathematical operations on vectors
x
y=c(3,2,4,3,7,6,1,1)
y
```

```
x+y
x+10
2*y
x*y
x/y
y^2

## Use of brackets (), [], {}
# these three brackets have different uses in R

# () is useful for grouping calculations
3*(x+1)

# [] is different -- this is used for indexing elements of a vector
# 2nd element of x
x[2]

# first five elements of x
x[1:5]

# all but the 3rd element of x
x[-3]

# 20th element of x doesn't exist
x[20]

# there are two logical values in R -- TRUE or FALSE
# we use a logical equal to see if 3 is equal to 4
3==4

# but is 3 less than 4?
3<4

# it is true that 3 is not equal (!=) 4
3!=4

# by using logical operators, we can extract parts of vectors
x;y

# we extract the values of x that are smaller than 40
x[x<40]

# since x and y are the same length, they can be matched, and
# we can find the values of y such that x is smaller than 40
y[x<40]



-----------------------------------
  R INTRODUCTION PART II
-----------------------------------

In this part, we will get introduced to R commands for inputting data and
outputting R output and graphs.

          scan command
          data.entry command
          read.table command
          saving output
          saving graphs

# There are several convenient ways of getting data into R.

# INPUTTING DATA FROM THE KEYBOARD:

# For small datasets, one can type the data directly into R using the scan command.
# Just type

mydata=scan()

# and enter the numbers, one or more on a line, and type a blank line when you are
# finished.

# Here's an example, entering in the temperatures 45, 55, 65, 23, 33.  I type
# temps at the end to verify that the data has been entered in correctly.

> temps=scan()
1: 45
2: 55 65
```

```
4: 23
5: 33
6:
Read 5 items
> temps
[1] 45 55 65 23 33

# Another convenient way of entering data is by the data.entry command, where you
# enter data by a spreadsheet format

# First, you have to define the variable name that will hold the data.  This indicates
# that mydata contains one value that is missing (R uses NA for missing).

mydata=c(NA)

# Then you can type

data.entry(mydata)

# You'll see a spreadsheet window.  You enter data into the spreadsheet (replace the NA
# entry with a number) and close the window when you are done.

# READING IN SOME DATA INTO R FROM A FILE

# For larger data files, I recommend creating them in some program outside of R
# and then reading them into R using the read.table command.

# Here is a simple example:  I have the following data that I wish to enter into R:
# (the number of wins and losses for five baseball teams)


TEAM               WINS     LOSSES
Kansas City        54       43
Minnesota 49       49
Chicago            49       50
Cleveland 41       58
Detroit            26       71

# STEP 1:  I entered the data into Excel, the first row containing the variable names
# TEAM, WINS, LOSSES.

# STEP 2:  I save the data as a "text, tab-delimited" file -- I call the file winslosses.txt

# STEP 3:  You have to place the file in a location where R can read it.  R has a "working # directory"
# where it is looking for files.  You can change the working directory
# to the folder which contains the data file.

# STEP 4:  You read the data into R using the read.table command.  Note that we indicate in # the command that
# there is a header line with the column names.

data=read.table("winslosses.txt",header=T)

# look at the data matrix
data

# You can also import data saved in .csv format using the read.csv command.
health_data=read.csv("health_exam_results.csv")

# You can also import directly from an internet location
data=read.table(url("http://www.uwlax.edu/faculty/toribio/data_sets/health_exam_results.txt"),header=T)


# SIMULATING VALUES FROM COMMON DISTRIBUTIONS
# From Normal distribuion
data_norm=rnorm(10000,mean=10,sd=2)


# SAVING R OUTPUT

# Here we describe several ways of saving R output.

# If you wish to save all or some of the output displayed in the R Console, then just
# select and copy the output and paste it into your favoriate word processor like Word.

# R also keeps track of all of the commands that you type in.  To save these commands,
# select Save History from the File menu and save this into a text file.

# You can also save graphs that are created using R using standard Copy commands.  We'll
# mention this again when we discuss graphing on R.
```

R INTRODUCTION PART III

In this part, we'll get some experience creating and manipulating
matrices in R. Also, we will be introduced to the notion of a data frame
which is a basic way of representing data in R. Also we'll create our first
R function.

Topics
- matrices
- accessing elements of matrices
- matrix computations
- row and column functions
- the apply() function
- data frames
- introduction to a function

MATRICES

In the last session, we were introduced to the following data that gives the number
of wins and losses for five baseball teams.

TEAM	WINS	LOSSES
Kansas City	54	43
Minnesota 49	49	
Chicago	49	50
Cleveland 41	58	
Detroit	26	71

We can create a matrix in R by use of the matrix command. The first argument is the
list of values, and the "nrow" and "ncol" options indicate the number of rows and columns
in the matrix. The values in the matrix are entered by columns -- the first five elements are placed in the
first column of the matrix and the second five values are placed in the second column.

```
wl=matrix(c(54,49,49,41,26,43,49,50,58,71),nrow=5,ncol=2)

wl
```

The above matrix isn't that useful since we don't have any labels attached to the rows
or columns. We can add labels by use of the dimnames command.

We first create a vector "teams" that contains the string labels for the five teams.

```
> teams=c("KC","Minn","Chi","Cle","Det")
```

Likewise, we create a vector "cols" that contains the string labels for the two columns
of the matrix.

```
> cols=c("Wins","Losses")
```

Then we can add these labels to the matrix "wl" by the dimnames command.

```
> dimnames(wl)=list(teams,cols)
> wl
     Wins Losses
KC     54     43
Minn   49     49
Chi    49     50
Cle    41     58
Det    26     71
>
```

ACCESSING ELEMENTS OF A MATRIX:

We access parts of a matrix by use of the notation

```
matrix_name[row numbers, col numbers]
```

To illustrate, here are some examples:

```
> wl[2,1] # the element in the 2nd row, 1st column
> wl[4,2] # the element in the 4th row, 2nd column
> wl[3,]  # the 3rd row of the matrix
  Wins Losses
    49     50
```

```
> wl[,2]  # the 2nd column of the matrix
  KC Minn  Chi  Cle  Det
  43   49   50   58   71

> wl[2:4,1]     # the submatrix consisting of the 2nd through 4th elements of the 1st column
Minn  Chi  Cle
  49   49   41
```

We can also refer to rows or columns by the names:

```
> wl["KC",]        # the row corresponding to "KC"
  Wins Losses
    54     43
> wl[,"Losses"]  # the "Losses" column
  KC Minn  Chi  Cle  Det
  43   49   50   58   71

>
```

MATRIX COMPUTATIONS

It is easy to perform computations using matrices. Here are some examples.

We define two 3 x 2 matrices A and B:

```
> A=matrix(1:6,nrow=3,ncol=2)
> A
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6


> B=matrix(c(2,4,6,8,10,12),nrow=3,ncol=2)
> B
     [,1] [,2]
[1,]    2    8
[2,]    4   10
[3,]    6   12
```

Multiply A by a constant and add a constant.

```
> 2*A+3
     [,1] [,2]
[1,]    5   11
[2,]    7   13
[3,]    9   15
```

Add two matrices.

```
> A+B
     [,1] [,2]
[1,]    3   12
[2,]    6   15
[3,]    9   18
```

Element by element multiplication of two matrices.

```
> A*B
     [,1] [,2]
[1,]    2   32
[2,]    8   50
[3,]   18   72
```

Element by element division of two matrices:

```
> A/B
     [,1] [,2]
[1,]  0.5  0.5
[2,]  0.5  0.5
[3,]  0.5  0.5
```

Transpose of a matrix

```
> t(A)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Matrix Multiplication

```
> t(A)%*%B
     [,1] [,2]
[1,]   28   64
[2,]   64  154
```

ROW AND COLUMN COMPUTATIONS

There are special commands that perform computations on each row or column of a matrix.

```
> rowSums(A)
[1] 5 7 9

> colSums(A)
[1]  6 15

> rowMeans(A)
[1] 2.5 3.5 4.

> colMeans(A)
[1] 2 5
```

THE APPLY FUNCTION

There is a very useful function, apply, for performing the operation of your choice on each row or each column of a matrix.

Say we are interested in finding the maximum value of each row of matrix A.

```
apply(A,1,max)
[1] 4 5 6
```

Or we wish to find the maximum of each column of A.

```
apply(A,2,max)
[1] 3 6
```

Generally the apply function has three arguments:

```
apply(matrix_name,dimension,function_name)
```

The dimension will be 1 for rows and 2 for columns.

So to find the median of each row of B, we type

```
apply(B,1,median)

[1] 5 7 9
```

DATA FRAMES

A data frame is a generalization of a matrix where the columns can be different types of data. For example, suppose we record the name, gender, and height for five students.

We put the names, genders, and heights in three vectors.

```
> names=c("Joe","Jill","Julie","Sam","Sue")
> gender=c("M","F","F","M","F")
> heights=c(70,67,63,67,66)
```

Then we use the data.frame command with arguments the three vectors.

```
>
> data=data.frame(names,gender,heights)
```

When we print data, we see that this command has put the variables together in a matrix format.

```
> data
  names gender heights
1   Joe      M      70
2  Jill      F      67
3 Julie      F      63
4   Sam      M      67
5   Sue      F      66
```

We extract variables from the data frame by the "$" suffix:

```
> data$heights
[1] 70 67 63 67 66
> data$gender
[1] M F F M F
Levels: F M
```

Many of the R functions will produce output that is the form of a data frame.

INTRODUCTION TO A FUNCTION

One nice feature of R is that you can define new commands by the use of functions.

Here is a simple example of a function. Suppose you want R to compute the mean absolute deviation (MAD) that is defined by

MAD=sum of abs(x - mean) / (number of observations)

Here is how we define a function -- the general syntax is

```
Name_of_function = function(x)
{
put your commands here
the output of the final command is the output of the function
}
```

Here we call the function MAD. The syntax looks like the following:

```
> MAD=function(x)
+ {
+ sum(abs(x-mean(x)))/length(x)
+ }
```

To execute our new function, we define a vector "data" and apply MAD to this vector:
The output is the mean absolute deviation of our data.

```
data=c(2,4,5,6,10)
> MAD(data)
[1] 2.08
```

R INTRODUCTION PART IV

In this part, we will meet some basic R commands for graphing data. Actually we will focus on a single command, plot, that will be your most useful tool.

- the plot command
- modifying plot elements
- adding elements to existing graph
- saving your graph

THE PLOT COMMAND

The basic graphing command is plot. To illustrate this, we look at exhibit 33 of chapter 5 from Tukey's EDA book which gives the median age of the U.S. for each of the census years between 1790 and 1950.

We put the year data in variable "year" and the median ages in "m.age".

```
> year=seq(1790,1950,10)
> m.age=c(15.9,15.7,15.9,16.5,17.2,17.9,19.5,20.2,20.6,21.6,
+ 22.9,23.8,24.9,26.1,27.1,29.5,30.4)
```

The basic command is

```
> plot(year,m.age)
```

which will produce a scatterplot of year (horizontal) against m.age (vertical).

There are other types of plots that can be produced -- they are accessed by the type = "letter" option of plot.

```
> plot(year,m.age,type="l")  # produces a line graph
> plot(year,m.age,type="b")  # points and lines between points
> plot(year,m.age,type="h")  # height bars (vertical)
```

```
> plot(year,m.age,type="o")  # overlaid points and connected lines
> plot(year,m.age,type="s")  # stairsteps
```

MODIFYING PLOT ELEMENTS:

Once a plot is constructed, you may be interested in
 - adding a title at the top of the graph
 - adding labels for the x and y axes
 - changing the limits of the horizontal and vertical axes
 - changing the plotting character
 - changing the thickness of the plotting line
 - changing the color of the line

All of these changes are done by adding options to the plot command
Here is a listing of the options:

```
TO ADD/CHANGE      OPTION IS
---------------------------------------
> # main title              main = "string"
> # x label                 xlab = "string"
> # y label                 ylab = "string"
> # limits of x axis        xlim=c(a,b)
> # limits of y axis        ylim=c(a,b)
> # plotting character      pch = "*"
> # line width              lwd = 1
> # line type               lty = 1
> # color                   col = 1
>
```

Below I experimented with all these options:

```
>
> plot(year,m.age,type="b",main="My graph",xlab="year",ylab="median age",
+ xlim=c(1750,2000),ylim=c(10,40),lwd=3,lty=2,col=3)
>
```

ADDING ELEMENTS TO EXISTING GRAPH

Here are useful commands for adding to the existing graph.

```
abline(a, b)                # this adds a line with intercept a and slope b
arrows(x1,y1,x2,y2)         # this adds an arrow from (x1,y1) to (x2,y2)
points(x,y)                 # adds points at the coordinates defined by vectors x and y
lines(x,y)                  # adds lines through the points given by x and y
text(xpos,ypos,text)        # adds text at the location (xpos,ypos)
title("title")     # adds a title to the graph
```

SAVING YOUR GRAPH

Once you have created a beautiful graph, you want to save it. There are many
ways of saving the graph into different formats, say postscript or jpeg. But a simple
way of saving the graph is to

 - copying the graph onto the clipboard
 - pasting the graph into a program like Word

Other examples:

```
data_norm=rnorm(10000,mean=10,sd=2)
hist(data_norm)
boxplot(data_norm)
qqnorm(data_norm)
qqline(data_norm)

data_pois=rpois(10000,lambda=4)
counts=tabulate(data_pois)
barplot(counts)
```

############## End of R session ############################