

A Quick Introduction to R

This document very briefly reviews the main features of the R statistical package. For full details, please refer to the various manuals and help files that come packaged with R, or the extra material referred to at the end of this document.

This document is in divided into five sections:

1. **Introduction:** How to download and install, overview of capabilities.
2. **R Basics:** Covers using R as a calculator, entering data, types of variables, random number generation, and getting help.
3. **Basic Statistics:** Covers t-tests with confidence intervals for means, chi-square tests with confidence intervals for proportions, non-parametrics, simple regression, generalized linear models including logistic regression.
4. **Graphics:** scatter plots, histograms, 3-D plots, boxplots
5. **Creating your own functions:** We will see here how to create our own functions, to extend the capabilities of R.

1 Introduction

1.1 What is R?

Quoted directly from the R WWW page (<http://www.r-project.org/>)

“R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

“R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice

for research in statistical methodology, and R provides an Open Source route to participation in that activity.

“One of R’s strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

“R is available as Free Software under the terms of the Free Software Foundation’s GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.”

In short: R is “free Splus”, a very sophisticated statistics and graphics package, used by most statisticians these days for modern analysis and statistical research.

2 Installation

Installation follows a few trivial steps:

1. Go to R web page, at <http://www.r-project.org/>
2. On the left hand side menu on the screen, click on “CRAN” which is under the “Download” item.
3. Pick a country site from which to download (for example, University of Toronto under Canada, but really you can pick any, all this effects is download speed).
4. Near the top of the page (e.g., <http://cran.stat.sfu.ca/>), inside the box labelled “Pre-compiled Binary Distributions”, pick the right file to download, depending on your operating system (e.g., click on “Windows (95 and later)” if you have any Windows system).
5. This brings you to a page where you select the part of R you need. While you may later want to download the set of user contributed functions, for now just click on “base”, which gets you the basic R program.
6. At the next screen, click on “R-2.3.1-win32.exe” (or similar). At this point you should be asked (via a prompt box) where you want to save the file. Pick a place on your computer to save this (e.g., `c:\temp`).
7. After file has downloaded, in your explorer (in Windows, use equivalent if you have another operating system), find where you have saved it, and click on “R-2.3.1-win32.exe” (or similar).

8. Follow directions on screen until program is installed. In general, you can just leave all options unchanged.
9. After installation is completed, click on the “R” icon on your desktop to begin the program. At this point, your screen should have an R prompt on it, and you are ready to begin using the program.

3 Adding Packages

You can easily extend R’s basic capabilities by adding in functions that others have created and made freely available through the web. Rather than surfing the web trying to find them, R makes adding these capabilities trivially easy by using the “Packages” menu items.

For example, to add a new package not previously downloaded onto your computer, click on “Packages → Install Package(s)”, pick a site from the list (usually a Canadian site) and choose the package you want to install from the long list (currently, about 900 packages are available). To activate this package, you then need to click on it in the ‘Packages → Load Package’ menu item. You are then ready to use it, and the help files for the package will also be automatically added to your installation. See, for example, the “Help → HTML help” menu item, and click on “Packages” in the browser window that opens.

4 R Basics

R can be used in many ways:

Use R as a Calculator:

```
> 2+3
[1] 5
> pi
[1] 3.141593
> 2+3*pi
[1] 11.42478
> log(2+3*pi)
[1] 2.435785
> exp(2.435785)
[1] 11.42478
```

```
> 6/7
[1] 0.8571429
> cos(pi)
[1] -1
```

Entering and Manipulating Data in R:

Scaler variables:

```
> a <- 3
> a
[1] 3
> b <- 5
> b
[1] 5
> b-a
[1] 2
> b/a
[1] 1.666667
```

Vector variables:

```
> x<-c(2,3,1,5,4,6,5,7,6,8)
> x
[1] 2 3 1 5 4 6 5 7 6 8
> y <- c(10, 12, 14, 13, 34, 23, 12, 34, 25, 43)
> y
[1] 10 12 14 13 34 23 12 34 25 43
> x[4]
[1] 5
> y[3]
[1] 14
> x[1:10]
[1] 2 3 1 5 4 6 5 7 6 8
> x[6:8]
[1] 6 5 7
> x[x > 5]
[1] 6 7 6 8
```

Functions on vectors:

```
> length(x)
[1] 10
> length(y)
[1] 10
> sum(x)
[1] 47
> sum(y)
[1] 220
> sum(x^2)
[1] 265
> mean(x)
[1] 4.7
> mean(y)
[1] 22
> var(x)
[1] 4.9
> var(y)
[1] 136.4444
> sqrt(var(x))
[1] 2.213594
> sqrt(var(y))
[1] 11.68094
> sum((x-mean(x))^2)
[1] 44.1
> summary(x)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.00   3.25   5.00   4.70   6.00   8.00
> summary(y)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
10.00  12.25  18.50  22.00  31.75  43.00
> summary(x^2)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.00  10.75  25.00  26.50  36.00  64.00
```

Matrices:

```
> z<- matrix(c(1,2,3,4,5,6,7,8,9), nrow=3, byrow=T)
> z
```

```

      [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> z.transpose<- matrix(c(1,2,3,4,5,6,7,8,9), nrow=3, byrow=F)
> z.transpose
      [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> z[2,3]
[1] 6
> z[,1]
[1] 1 4 7
> z[1,]
[1] 1 2 3
> z[3,]
[1] 7 8 9

```

You can also have character variables:

```

> sex <- c("male", "female", "female", "male", "male", "female", "female")
> sex
[1] "male"  "female" "female" "male"   "male"   "female" "female"
> smoking <- c("yes", "no", "no", "yes", "no", "yes", "no", "no", "yes", "no")
> smoking
[1] "yes" "no"  "no"  "yes" "no"   "yes" "no"  "no"  "yes" "no"

```

You can declare these variables to be factor variables for regression analyses:

```

> sex.factor<- as.factor(sex)
> sex.factor
[1] male  female female male   male   female female
Levels: female male

```

It is trivially easy to generate random numbers in R, useful for all kinds of statistical analyses and simulations:

```

> x<-rnorm(10, mean=3, sd=2)

```

```

> x
[1] 2.135011 7.573118 3.840814 5.544555 3.393277 2.731562
[7] 1.780588 4.576343 3.250077 3.908082
> y<-rbinom(10, size=1, prob=0.4)
> y
[1] 0 1 1 1 0 0 1 0 1 0
> z<- 5 + 2*x + 3*y +rnorm(10, mean=0, sd=0.5)
> z
[1] 9.44732 23.30763 15.34688 19.62479 11.30383 10.89258
[7] 11.22488 13.33018 15.11767 12.81894

```

Data Frames:

Most data sets are entered into R as data frames. To create a data frame, you can enter:

```

> my.data.frame<-data.frame(x,y,z)
> my.data.frame
      x y      z
1 2.135011 0 9.44732
2 7.573118 1 23.30763
3 3.840814 1 15.34688
4 5.544555 1 19.62479
5 3.393277 0 11.30383
6 2.731562 0 10.89258
7 1.780588 1 11.22488
8 4.576343 0 13.33018
9 3.250077 1 15.11767
10 3.908082 0 12.81894

```

To get help on any R function, simply type

```
> help(sum)
```

and a help window will pop up, in this case for the function “sum”. R also comes with extensive manuals, html help files, a nice introduction, etc. Extremely Useful Suggestion for this Course: Look at the help files for read.table to see how to enter data from a text file on your computer.

5 Basic Statistical Routines

You can use R to do any of the standard statistical routines you will typically need. R includes everything you will learn in 607/621, and many other routines, with more added regularly. R has many more modern (frequentist) routines available compared to other packages such as SAS or STATA, and whatever is not in the base package is likely to be in the contributed section. In addition, whatever is not available in either of these sources, “built-in” routines can easily be added on your own, as we will soon see.

T-Tests

```
> group1 <- x[y==1]
> group0 <- x[y==0]
> group1
[1] 7.573118 3.840814 5.544555 1.780588 3.250077
> group0
[1] 2.135011 3.393277 2.731562 4.576343 3.908082
> t.test(group1, group0)
```

Welch Two Sample t-test

```
data: group1 and group0 t = 0.9667, df = 5.431, p-value = 0.3748
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.675244 3.773195
sample estimates: mean of x mean of y
 4.397831 3.348855
```

Chi-square tests

Suppose you have a two-by-two table of data, such as:

	Patient Independent	Patient Dependent
Stroke Unit	67	34
Medical Unit	46	45

You can simply enter:

```
> prop.test(c(67,46), c(67+34, 46+45) )
```

2-sample test for equality of proportions with continuity correction

```
data: c(67, 46) out of c(67 + 34, 46 + 45) X-squared = 4.2965, df
= 1, p-value = 0.03819 alternative hypothesis: two.sided 95
percent confidence interval:
 0.009420794 0.306322868
sample estimates:
  prop 1    prop 2
0.6633663 0.5054945
```

More generally:

```
> smokers <- rbinom(100, 1, 0.2)
> non.smokers<-rbinom(200, 1, 0.12)
> prop.test(c(sum(smokers), sum(non.smokers)), c(100,200), correct=F)
```

2-sample test for equality of proportions without continuity correction

```
data: c(sum(smokers), sum(non.smokers)) out of c(100, 200)
X-squared = 2.1319, df = 1, p-value = 0.1443 alternative
hypothesis: two.sided 95 percent confidence interval:
 -0.02659294 0.15659294
sample estimates: prop 1 prop 2
 0.200 0.135
```

Linear Models

Recall our data frame:

```
> my.data.frame
      x y      z
1 2.135011 0 9.44732
2 7.573118 1 23.30763
3 3.840814 1 15.34688
4 5.544555 1 19.62479
5 3.393277 0 11.30383
6 2.731562 0 10.89258
7 1.780588 1 11.22488
8 4.576343 0 13.33018
9 3.250077 1 15.11767
10 3.908082 0 12.81894
```

Suppose we want simple linear regression model of z on x and y .

```
> lin.reg<-lm(z ~ x + y)
> summary(lin.reg)
```

Call:

```
lm(formula = z ~ x + y)
```

Residuals:

Min	1Q	Median	3Q	Max
-0.6821	-0.4339	0.0893	0.3849	0.5679

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	4.8644	0.4245	11.460	8.66e-06	***
x	1.9989	0.1064	18.792	3.00e-07	***
y	3.2690	0.3450	9.475	3.05e-05	***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5162 on 7 degrees of freedom

Multiple R-Squared: 0.9889, Adjusted R-squared: 0.9857

F-statistic: 311.6 on 2 and 7 DF, p-value: 1.444e-07

Recall that z was generated by

```
> z<- 5 + 2*x + 3*y +rnorm(10, mean=0, sd=0.5)
```

so our estimates are close to expected (including sd estimate).

Other types of regression equations are available through generalized linear models:

For logistic regression, type (assuming z is dichotomous)

```
> glm(z ~ x + y, family=binomial(link="logit"))
```

For poisson regression, type (assuming z is a count variable)

```
> glm(z ~ x + y, family=poisson(link = "log"))
```

and so on.

6 Graphics

R is capable of producing very high quality graphics in a variety of formats (wmf, ps, jpg, pdf, png, bmp).

For a scatter plot, simply type:

```
> plot(x,z)
```

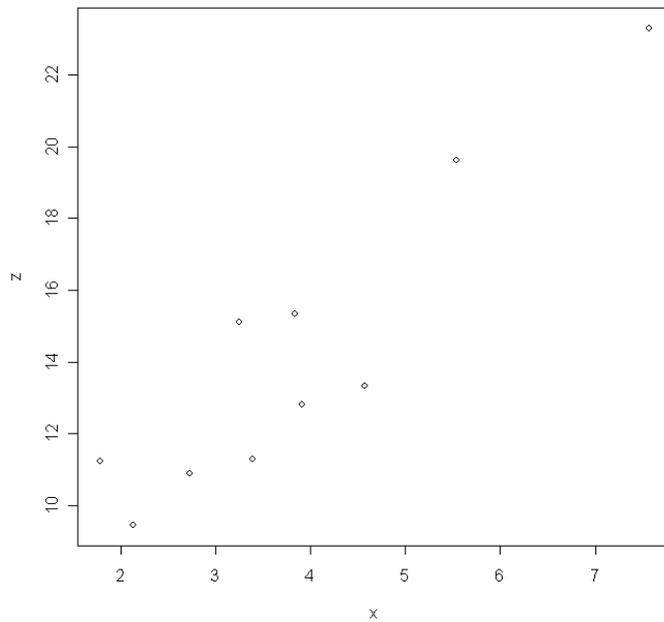


Figure 1: A simple scatter plot

Adding titles and labels for each axis is easy:

```
> plot(x,z, main="Scatter plot of age versus outcome", xlab="age", ylab="outcome")
```

Adding the best fitting regression line to the plot as well:

```
> plot(x,z, main="Scatter plot of age versus outcome", xlab="age", ylab="outcome")  
> lsfit(x,z)$coef
```

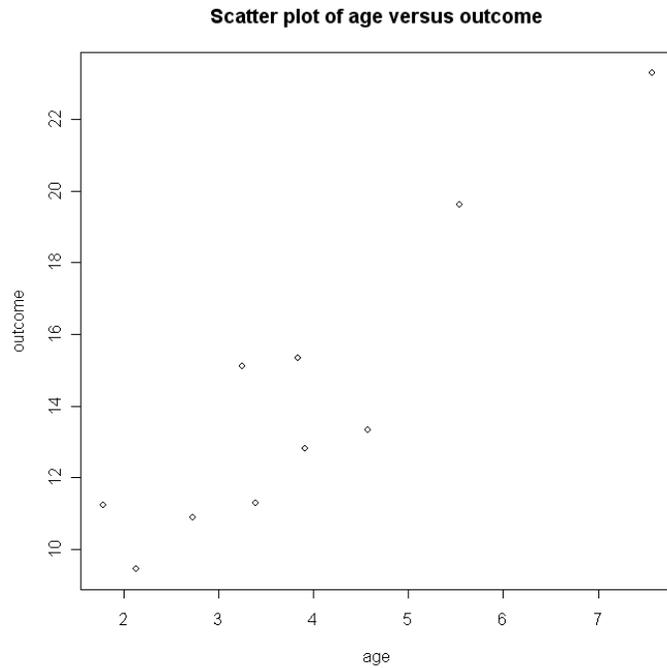


Figure 2: A simple scatter plot with labels and title

```

Intercept      X
 5.236471  2.324865
> age.range <- seq(2,7,by=0.01)
> outcome.fit <- 5.236471 + 2.324865*age.range
> points(age.range, outcome.fit, type="l")

```

You can plot more than one curve on a single plot, and label them via a legend:

```

> range <- seq(-10, 10, by = 0.001)
> norm1 <- dnorm(range, mean=0, sd=1)
> norm2 <- dnorm(range, mean=1, sd=2)
> plot(range, norm1, type="l", lty=1, main="Two Normal Distributions",
       xlab="Range", ylab="Probability Density")
> points(range, norm2, type="l", lty=2)
> legend(x=-10, y=0.4, legend= c("N(0,1)", "N(1,2)"), lty=c(1,2))

```

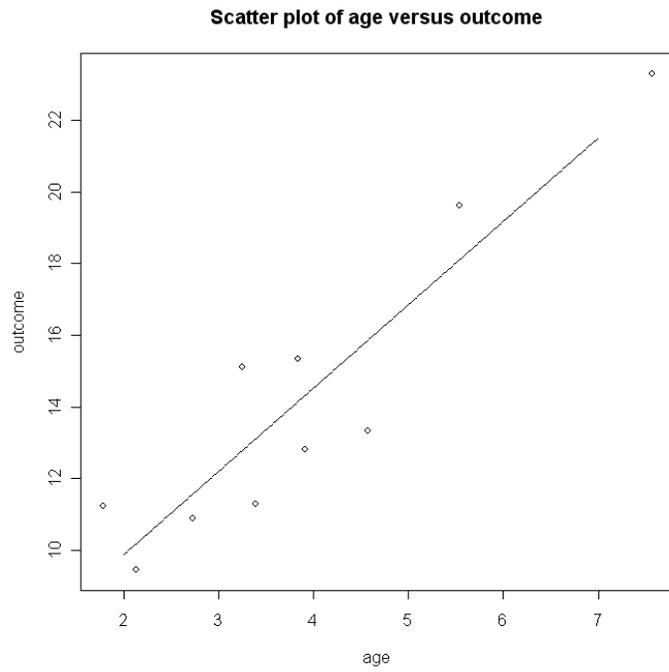


Figure 3: Scatter plot with best fitting line added

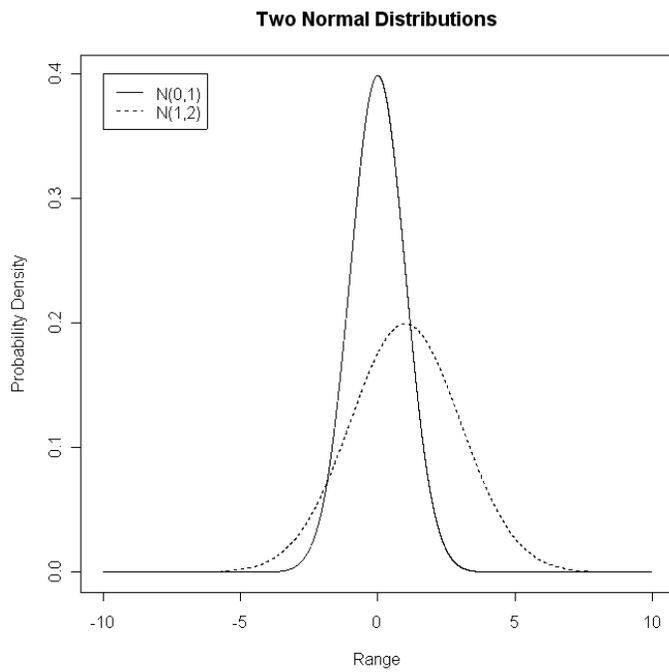


Figure 4: Distributions with legend

You can also do histograms ...

```
> hist(rnorm(1000,mean=0, sd=1), main = "Sample From N(0,1)
      Distribution", xlab = "Range", ylab="Frequency")
```

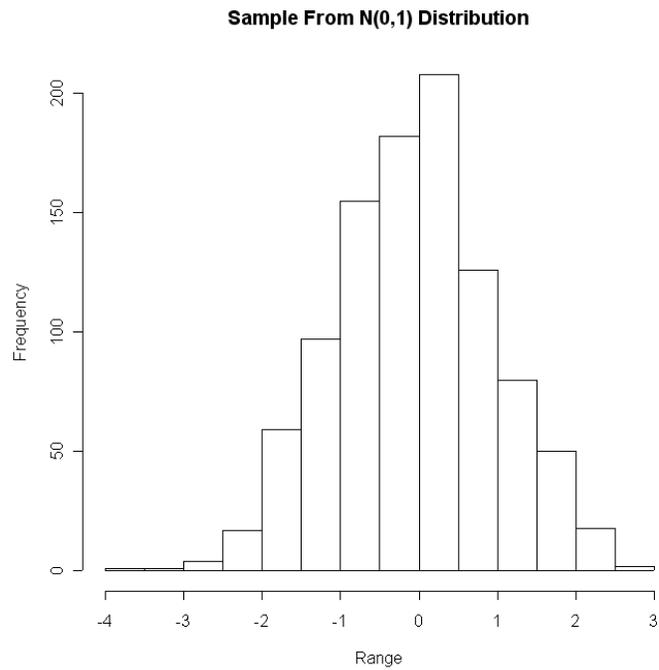


Figure 5: Histogram

... and boxplots.

```
> boxplot(rnorm(1000, mean=0, sd=1), rnorm(1000, mean=1, sd=2),
          names=c("N(0,1)", "N(1,2)"))
```

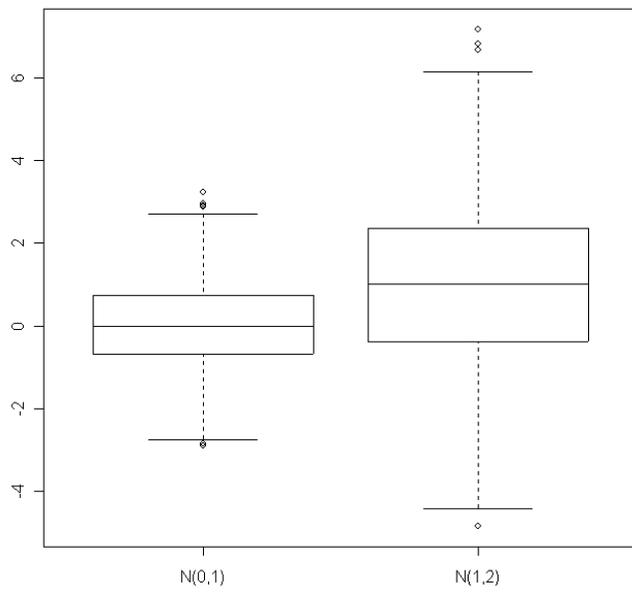


Figure 6: Boxplot

7 Writing your Own Functions

It is very easy to write your own functions in R. Once these are written, they become permanent functions in your version of R, available just like any other function. Here I present two examples, one easy, one (slightly) more complex.

First the easy example, creating a function to add two numbers. Of course, this is a silly example, since R already does this easily, but its simplicity will allow us to see how functions are constructed in general.

```
> add.2.numbers <- function(a,b)
{
  sum = a+b
  return(sum)
}
> add.2.numbers(3,4)
[1] 7
> pi
[1] 3.141593
> exp(1)
[1] 2.718282
> add.2.numbers(pi, exp(1))
[1] 5.859874
```

That is all one needs to do to make a function!

Now for something more complex, we will create a function to carry out Bayesian inferences for a single normal mean, when the variance is assumed known.

The theory is the following (to remind you of what we covered last class):

Suppose that your data set, $X = (x_1, x_2, \dots, x_n)$, with sample size n is known to arise from a Normal distribution, $x_i \sim N(\mu, \sigma^2)$, $i = 1, 2, \dots, n$, i.e., the likelihood function for the data is normal. Suppose also that the variance of the data, σ^2 is a known constant, and so the only unknown parameter to be estimated is the mean. Suppose that the prior distribution for the unknown mean μ is $N(\theta, \tau^2)$.

If we combine the information in the prior distribution with the information in the data set via Bayes Theorem, we derive the posterior distribution, which summarizes current information about the mean μ . It happens that the posterior is also normal, and the exact formula

is:

$$\mu_{posterior} \sim N \left(\frac{\frac{\theta}{\tau^2} + \frac{n\bar{x}}{\sigma^2}}{\frac{1}{\tau^2} + \frac{n}{\sigma^2}}, \left[\frac{1}{\tau^2} + \frac{n}{\sigma^2} \right]^{-1} \right),$$

where \bar{x} is the mean of the observed data, and $[\cdot]^{-1}$ indicates matrix inversion.

A function that implements this formula in R is:

```
> post.normal.mean <- function(x, prior.mean, prior.var, data.var)
{
#####
# R function for Bayesian analysis of normal mean, variance known #
# Parameters included are: #
# # #
# Inputs: #
# # #
# x = vector of data #
# prior.mean = prior mean #
# prior.var = prior variance #
# data.var = assumed known variance of data #
# # #
# Outputs: #
# # #
# post.mean = posterior mean #
# post.var = posterior variance #
# # #
#####

n<- length(x)
x.bar <- mean(x)
post.mean.numerator <- prior.mean/prior.var + n*x.bar/data.var
post.mean.denominator <- 1/prior.var + n/data.var
post.mean <- post.mean.numerator/post.mean.denominator
post.var <- (1/(1/prior.var + n/data.var))

a <- "Post mean = "
b <- "Post Var = "

cat(a, post.mean, ",", b, post.var, "\n")
}
> post.normal.mean(1:10, 0, 1000, 1)
Post mean = 5.49945 , Post Var = 0.09999
```

Another useful way to create output in functions is illustrated below. This format is especially useful if you want to use the program's results later on in your R session.

```

> post.normal.mean <- function(x, prior.mean, prior.var, data.var)
{
#####
# R function for Bayesian analysis of normal mean, variance known #
# Parameters included are: #
# # #
# Inputs: #
# # #
# x = vector of data #
# prior.mean = prior mean #
# prior.var = prior variance #
# data.var = assumed known variance of data #
# # #
# Outputs: #
# # #
# post.mean = posterior mean #
# post.var = posterior variance #
# # #
#####

n<- length(x)
x.bar <- mean(x)
post.mean.numerator <- prior.mean/prior.var + n*x.bar/data.var
post.mean.denominator <- 1/prior.var + n/data.var
post.mean <- post.mean.numerator/post.mean.denominator
post.var <- (1/(1/prior.var + n/data.var))

posterior.parameters <- list(post.mean= post.mean, post.var = post.var)
return(posterior.parameters)
}
> post.normal.mean(1:10, 0, 1000, 1)
$post.mean
[1] 5.49945

$post.var
[1] 0.09999

```

A useful function relating to functions is `args`, which reminds you of the required arguments of the function. For example:

```

> args(post.normal.mean)
function (x, prior.mean, prior.var, data.var)

```

8 Conclusion

R is a very powerful package, we have just scratched the surface of what is available. Its interactive nature makes it more pleasant to use than “batch” programs like SAS, and its developmental environment makes it the first choice of many, if not the majority, of statisticians today.

Some other resources include:

- A useful book titled “Analysis of Epidemiological Data Using R and Epicalc,” available at

`http://cran.r-project.org/doc/contrib/Epicalc_Book.pdf`

- A series of useful tips are available at

`http://pj.freefaculty.org/R/statsRus.html`

I would be happy to answer any questions you have about R, just email me.