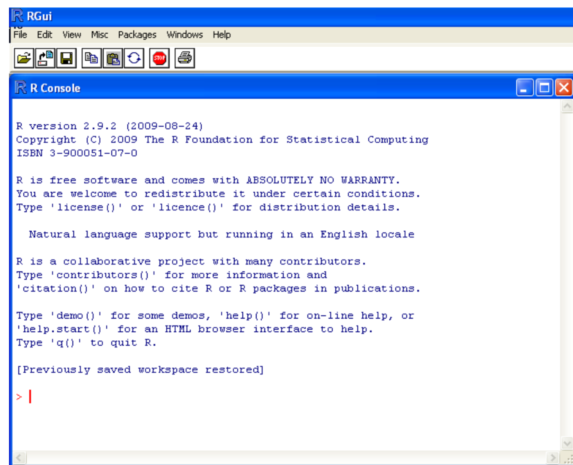
Getting started with R

This handout gives a very brief introduction to R to give you a rough idea of some basic R features.

Start up R

Start R by double-clicking on the R shortcut on the Desktop. A window called the R **Console** will open up.



Type commands after the prompt > and then press the <**ENTER**> key.

Note: anything after the pound symbol # is a comment—explanatory text that is not executed.

```
>  4*9    # Simple arithmetic
[1] 36
```

If you strike the <ENTER> key before typing a complete expression, you will see the *continuation prompt*, the plus sign (+). For example, suppose you wish to calculate $3 + 2 * (8 - 4)$, but you accidentally strike the <ENTER> key after typing the 8:

```
>  3+2*(8 <ENTER>
+
```

Finish the expression by typing -4) after the +

```
+ - 4) <ENTER>
[1] 11
```

Create a sequence incrementing by 1:

```
> 20:30
[1]  20 21 22 23 24 25 26 27 28 29 30
```

```
> 2.5 :  -3.5
[1] 2.5 1.5 0.5 -0.5 -1.5 -2.5 -3.5
```

We will create an object called `dog` and assign it the values 1, 2, 3, 4, 5. The symbol <- is the assignment operator.

```
> dog <- 1:5

> dog
[1] 1 2 3 4 5

> dog + 10
[1] 11 12 13 14 15

> 3*dog
[1] 3 6 9 12 15

> sum(dog)    # 1+2+3+4+5
```

The object `dog` is called a *vector*.

If you need to abort a command, type the escape key <**ESC**>.

To obtain help on any of the commands, type the name of the command you wish help on:

```
> ?hist
```

Importing a data set

Download the file **States03.csv** on the web page. You will need to place it in your *working directory*

```
> getwd()
```

Once you've copied **States03.csv** data to this directory, read in the file **States03** using the `read.csv` command. We will store it in an object (a **data frame**) called **States03**. This data set contains information about the 50 states from various government sources (from 2003).

```
> States03 <- read.csv("States03.csv")
```

To see the data in an editor:

```
> edit(States03)
```

Modify values by typing directly into the editor.

To exit the edit window, press the **ESC** key.

To view the names of the variables in **States03**:

```
> names(States03)
```

View the first part of the data by using the `head` command:

```
> head(States03)
```

The columns are the *variables*. There are two types of variables: *numeric*, for example, `Pop18` (percentage of population under 18 years old) and `PropertyCrime` (property crime per 100000) and *factor* (also called *categorical*), for example `Region` and `DeathPenalty`. The rows are called *observations* or *cases*.

To check the size (dimension) of the data frame, type

```
> dim(States03)
[1] 50 24
```

This tells us that there are 50 rows and 24 columns.

Tables, bar charts and histograms

The factor variable `DeathPenalty` in the **States03** data set assigns each state to one of two *levels*: Yes or No.

```
> head(States03$DeathPenalty)   #First six rows
[1] Yes No  Yes Yes Yes Yes
Levels: No Yes
> data.class(States03$DeathPenalty)
[1] "factor"
> table(States03$DeathPenalty)
  No Yes
  12  38
```

Remarks:

- The $ is one way to access the variables of a data frame. Another option, which we will learn later, is to create a new vector object by subsetting.
- `R` is *case-sensitive*! `DeathPenalty` and `deathPenalty` are considered different.

To visualize the distribution of a factor variable, create a *bar chart*:

```
> barplot(table(States03$DeathPenalty))
```

Compare two categorical variables, the region and whether or not a state has the death penalty:

```
> table(States03$Region, States03$DeathPenalty)
            No Yes
  Midwest    5   7
  Northeast  4   7
  South      1  13
  West       2  11
```

To see the distribution of a numeric variable, create a histogram of the populations in the 50 states.

```
> hist(States03$Pop)
```

The shape of the distribution of this variable is *right-skewed*.

Remark The variables in a data frame are not immediately accessible for use:

```
> Pop
Error: object 'Pop' not found
```

If you use a variable frequently, you may want to extract it and store it in a vector:

```
> Pop <- States03$Pop
> Pop
[1]  4500.752   648.818  5580.811  2725.714 35484.453
[6]  4550.688  3483.372   817.491 17019.068  8684.715
...
```

Numeric Summaries

To find the mean and median of a variable:

```
> mean(Pop)
```

```
> median(Pop)
```

In addition, we can find the range, or just the maximum and minimum of a variable:

```
> range(Pop)
```

```
> max(Pop)
```

```
> min(Pop)
```

We can find which observation corresponds to the one with the highest population:

```
> which(Pop==35484.453)
[1] 5
```

The observation in the 5th row is California.

To find the sample variance or the sample standard deviation:

```
> var(Pop)
```

```
> sd(Pop) # standard deviation
```

To find the *quartiles*,

```
> quantile(Pop)
```

The `tapply` command allows you to compute numeric summaries on values based on levels of a factor variable. For instance, find the mean population by region,

```
> tapply(Pop, States03$Region, mean)
```

or the median violent crime rate grouped by whether or not a state has the death penalty:

```
> tapply(States03$ViolentCrime, States03$DeathPenalty, median)
```

Boxplots

Boxplots give a visualization of the 5-number summary of a variable.

```
> boxplot(Pop)
> boxplot(Pop ~ Region, data = States03)  #population grouped by region
```

The `boxplot` command offers the option of using a *formula* syntax. Here, since we can specify the data set (**States03**), we don't have to use the $ to access the variables.

Misc. Remarks

- Functions in `R` are called by typing their name followed by arguments surrounded by *parentheses*: ex. `hist(Pop)`. Typing a function name without parentheses will give the code for the function.

  ```
  > sd
  ```

- We saw earlier that we can assign names to data (we created a vector called `dog`.) Names can be any length, must start with a letter, and may contain letters or numbers:

  ```
  > fish25<- 10:35
  > fish25
  ```

  Certain names are **reserved** so be careful to not use them: `cat`, `c`, `t`, `T`, `F`,...

  To be safe, before making an assignment, type the name:

  ```
  > whale
  [1] Problem:  Object "whale" not found
  ```

  Safe to use `whale`!

  ```
  > whale <- 200
  > objects()
  > rm(whale)
  > objects()
  ```

- In general, `R` is space-insensitive.

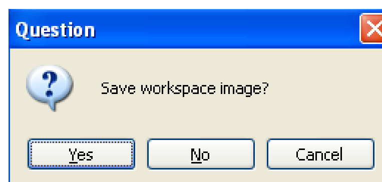
  ```
  > 3 +4
  > 3+    4
  > mean(3+5)
  > mean ( 3 + 5 )
  ```

  BUT, the assignment operator must not have spaces! `<-` is different from `<   -`

- To quit, type

  ```
  > q()
  ```

  You will be given an option to **Save the workspace image**.



  If you select **Yes**, then all objects created in this session are saved in your working directory so that the next time you start up `R`, these objects will still be available. You will not have to re-import **States03**, for instance.

  You can, for back-up purposes, save data to an external file/disk by using, for instance, the `write.csv` command. See the help file for more information.

Optional

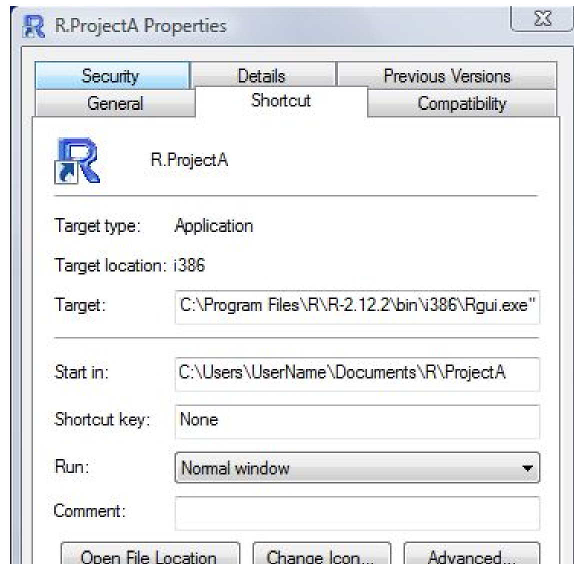
Windows You can have R start automatically in a particular working directory by modifying the R short-cut.

Suppose you keep all your R projects in a folder called R. Suppose you have a particular project, **Project A**. Create the folder **ProjectA** as a subfolder of the folder R. On the desktop of your PC, find the R shortcut, right-click and select **Properties**. In the **Start In:** field, type the path to the folder you want to use as the working directory. For instance,

```
"C:\Users\YourName\Documents\R\ProjectA"
```

or on Windows XP, perhaps,

```
"C:\Documents and Settings\MyName\My Documents\R\ProjectA
```



and then click **OK**.

Starting R by clicking on this short-cut will automatically load the **ProjectA** working directory.

The exact path above may vary, depending on the version of your operating system or where you store your own documents.

Macintosh To have R automatically start with **Project A** being the working directory, drag the **ProjectA** folder onto the alias for R .

Vectors in R

The basic data object in R is the vector. Even scalars are vectors of length 1.

There are several ways to create vectors.

The : operator creates sequences incrementing/decrementing by 1.

```
> 1:10
> 5:-6
```

The `seq()` function creates sequences also.

```
> seq(0,3,by=.2)
> seq(0,3,length=15)
```

To create vectors with no particular pattern, use the `c()` function (`c` for **c**ombine).

```
> c(1,4,8,2,9)
> x <- c(2,0,-4)
> x
> c(x, 0:5, x)
```

For longer vectors, use `scan()`.

```
> y <- scan() <ENTER>
1:  3 5 8 0 -2 4 <ENTER>
7:  4 11 8 0 <ENTER>
11:  0 3 <ENTER>
13:  <ENTER>
Read 12 items
> y
```

For vectors of characters,

```
> c("a","b","c","d")
```

or logical values (note that there are no double quotes):

```
> c(T,F,F,T,T,F)
```

The `rep()` command for repeating values:

```
> rep("a",5)
> rep(c("a","b"),5)
> rep(c("a","b"),c(5,2))
```

Basic Arithmetic

```
> x <- 1:5
```

```
> x-3
> x*10
> x/10
> x^2
> 2^x
> w <- 6:10
> w
> x*w
```

Note that multiplication is being performed coordinate-wise.

```
> x < 3
> x == 4
```

Subsetting

In many cases, we will want only a portion of a data set. For subsetting a vector, the basic syntax is `vector[`*index*`]`. In particular, note the use of *brackets* to indicate that we are subsetting.

```
> z <- c(8,3,0,9,9,2,1,3)
```

The fourth element of `z`:

```
> z[4]
```

The first, third and fourth element,

```
> z[c(1,3,4)]
```

All elements *except* the first, third and fourth:

```
> z[-c(1,3,4)]
```

The elements of `z` in reverse:

```
> z[8:1]
```

The `subset` command

We mentioned earlier that one way to just extract a single variable from a data frame is to use the $:

```
> Income <- States03$Income
> head(Income)
```

Another option is to use the `subset` command; this command can also extract subsets that satisfy certain conditions. The basic syntax is `subset(data, select=columns, subset=row.condition)`. The row condition must be a logical statement (something that evaluates to TRUE/FALSE.)

Suppose you wish to extract income for all states except the south:

```
> Income2 <- subset(States03, select==Income, subset=Region!="South", drop=TRUE)
```

```
> mean(Income2)
```

The `drop=TRUE` argument is needed to produce a vector. For instance, compare

```
> subset(States03, select=Pop18, drop=TRUE)
```

to

```
> subset(States03, select=Pop18)
```

The first is a vector, the second a data frame.

To subset the income for West and Midwest only

```
> Income3 <- subset(States03, select=Income, subset=Region="West" | Region=="Midwest",
drop=TRUE)
```

The vertical bar stands for "or."

Suppose you want to find those observations when the income was greater than the mean income. We'll store this in a vector called `index`.

```
> index <- which(Income > mean(Income))

> head(index)
[1]  2  5  6  7  8 11
```

Thus, observations in rows 2, 5, 6, 7, 8, 11 are the first six that correspond to states that had income more than the average income (average over all 50 states).

```
> States03$State[index]
 [1] Alaska        California    Colorado      Connecticut
 ...
50 Levels: Alabama Alaska Arizona Arkansas California Colorado

> data.class(States03$State[index])
[1] "factor"

> as.character(States03$State[index])
 [1] "Alaska"        "California"    "Colorado"      "Connecticut"
 ...

 > data.class(as.character(States03$State[index]))
[1] "character"
```

Vectorized operators

Operator	Meaning
==	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
&	vectorized AND
|	vectorized OR
!	not

Programming Note: The vectorized AND and OR are for use with vectors (when you are extracting subsets of vectors). For control in programming (ex. when writing `for` or `if` statements), the operators are && and ||.

Misc. commands on vectors

```
> length(x)    # number of elements in x

> sum(x)   #add elements in x

> sort(y) # sort in increasing order

> sample(y) # permute y

> sample(y, 2) # random sample of size 2 from elements in y
```

Data Frames in R

Most data will be stored in data frames, rectangular arrays which usually are formed by combining columns of vectors. **States03** is an example of a data frame.

```
> data.class(States03)
```

Subsetting data frames

For subsetting a data frame, use the syntax
`data[row.index, column.index]`.

For instance, row 5, column 3:

```
> States03[5,3]
```

or rows 1 through 10, columns 1 and 3:

```
> States03[1:10, c(1,3)]
```

or all rows *except* 1 through 10, and keep columns 1 and 3:

```
> States03[-(1:10), c(1,3)]
```

However, for more complicated subsets, use the `subset` command.

Create a subset of just the western states:

```
> StatesWest <- subset(States03, subset=Region=="West")
> head(StatesWest)
```

Create a subset of just the western states data and columns 1, 6, 7:

```
> StatesWest<- subset(States03, select=c(1,6,7), subset=Region=="West")
> head(StatesWest)
```

Creating a data frame

Use the `data.frame` command to bind two more more vectors (variables) together by column. For instance,

```
> x <- seq(0,1,length=10)
> x
> y <- rep(c("a","b"),5)
> y
> my.data <- data.frame(X=x,Y=y)
> my.data
```

Add another column to a data frame:

Method 1

```
> my.data <- cbind(my.data, Z=1:10)
```

`cbind()` for *column*-binding.

```
> my.data
> dim(my.data) # dimension of data frame
```

Method 2

Suppose you want the new column to be $\log(Z)$, which we'll call `logZ`:

```
> my.data$logZ <- log(my.data$Z)
> my.data
```