Using

R

Introductory Guide

University of Toronto Mississauga Library
Hazel McCallion Academic Learning Centre

FURTHER ASSISTANCE

If you have questions or need assistance,
please contact:

Andrew Nicholson
GIS/Data Librarian
U of T Mississauga Library – Rm. 360
Hazel McCallion Academic Learning Centre
905-828-3886
andrew.nicholson@utoronto.ca

Tanya Kenesky
GIS/Data Technician
U of T Mississauga Library – Rm. 360
Hazel McCallion Academic Learning Centre
905-569-4525
tanya.kenesky@utoronto.ca

OR

Drop into the
AstraZeneca Canada
Centre for Information & Technological Literacy
U of T Mississauga Library – Rm. 360
Hazel McCallion Academic Learning Centre

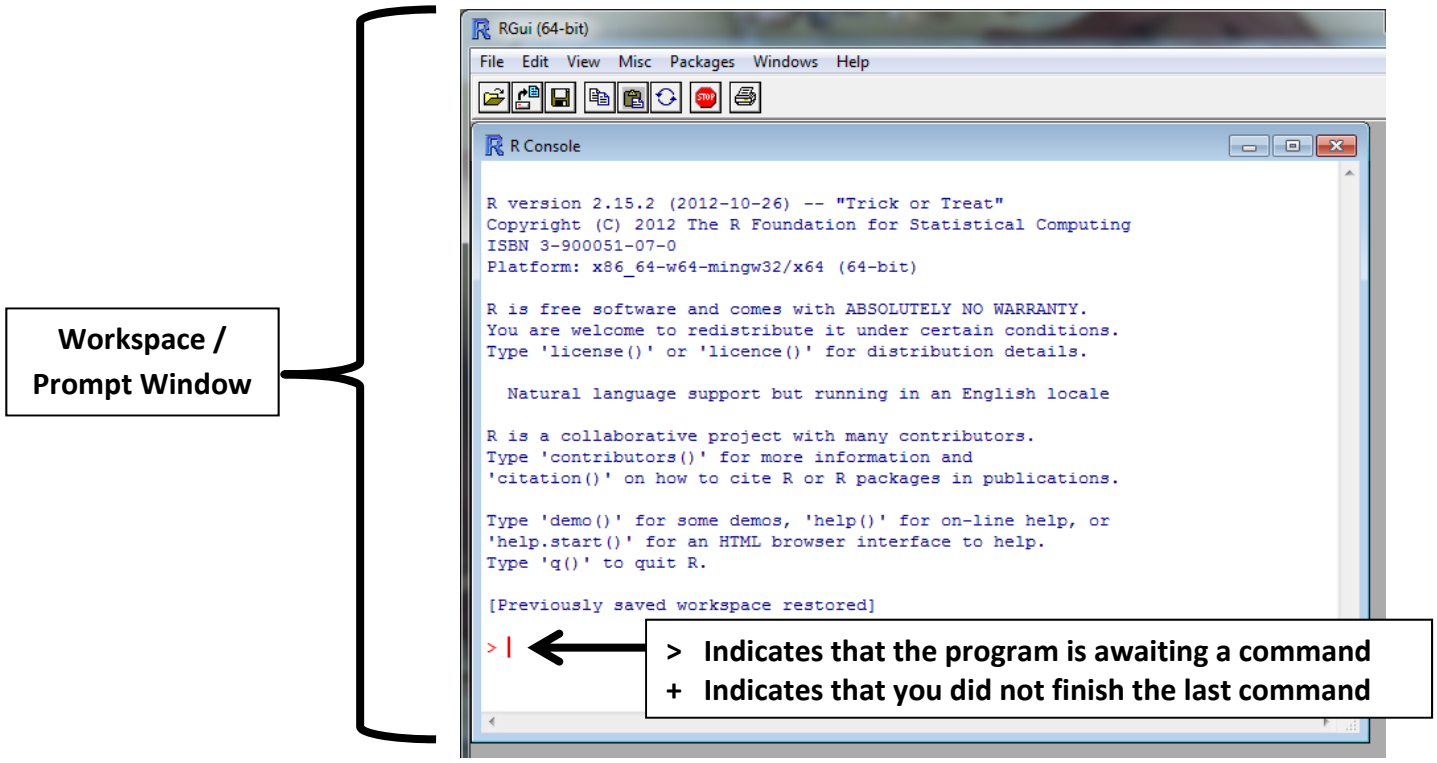
Between 9am & 5pm
Monday to Friday

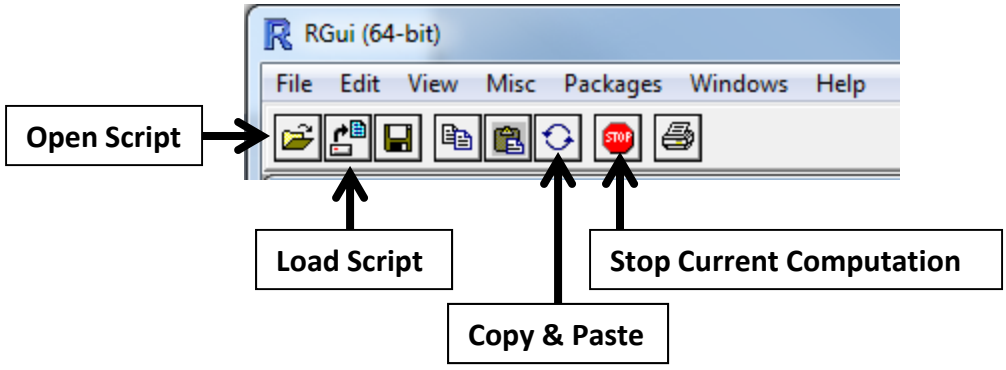
A Basic Introduction to R

Table of Contents

Introduction to R

R is a software and language where users can process statistical analysis. The **prompt window** (image below) is the main interface that you will be using to input data.





Familiarizing yourself with R

Setting the Working Directory

Setting the working directory will allow you to save items to the folder of your choosing without having to later browse for it. You can set your working directory by:

1) Using your mouse, select **File > Change dir…**
2) Typing in the command:

> setwd ("type in the path name")

```
> setwd("C:/[redacted]/R")
```

Please note that in windows you would use a / (forward slash) rather than a \ (backslash) when specifying the path name.

Saving the Workspace Image

By saving your workspace image you are saving all the objects contained within it, not the commands that you have previously run.

You can save your workspace in a few ways.

1) Using your mouse, select **File > Save Workspace…**
2) Typing in the command

```
> save.image()
```

You can name your saved workspace image by changing the name in the command. Be sure to include the **.RData** file extension.

```
> save.image("new_name.RData")
```

3) By quitting R. The program will ask if you would like to save your workspace image. Here you have the option of saving, not saving, or cancelling the action.

Loading the Workspace

You can load your workspace by:

1) Using your mouse, select **File > Load Workspace…**
2) Typing in the command

```
> load("new_name.RData")
```

Saving and Loading History

By saving your history you are saving all the commands that you have entered into R.

You can save and load your history by:

1) Using your mouse, select **File > Save History…** or **File > Load History…**
2) Typing in the command

```
> savehistory("new_name.Rhistory")
```

or

```
> loadhistory("new_name.Rhistory")
```

Importing Data

Often there are sample data sets or data packages available that can be imported into R for viewing. You can import a data package from software such as Excel, Minitab, and SPSS.

Excel

For this examples let assume the Excel data package is called **dpackage** and that the data contained within the package is called **thedata** and the new file we are creating is called **mydata**.

Step 1 – import the data package `> library(dpackage)`
Step 2 – loading the data from the package `> mydata <- read.xls("thedata")`

Minitab File

If the data is in a Minitab Worksheet it can be read with the **foreign** package that is in the core R library. In this example let assume that the minitab data file is called **mtabdata** and the new file we are creating is called **mydata**.

Step 1 – import the data package `> library (foreign)`
Step 2 – load the data `> mydata <- read.mtp("mtabdata.mtp")`

SPSS File

If the data is in a SPSS format it can be read with the **foreign** package that is in the core R library. In this example let assume that the SPSS data file is called **SPSSdata** and the new file we are creating is called **mydata**.

Step 1 – import the data package `> library (foreign)`
Step 2 – load the data `> mydata <- read.spss("SPSSdata.spss")`

Quitting R

To quit R use the command `> q()` or `> quit()`. You can also use the drop down menu **File > Exit**.

Basic Operations

In R you are able to process simple arithmetic and comparisons.

Arithmetic		Comparison	
addition	+	less than	<
Subtraction	-	greater than	>
multiplication	*	less than or equal to	<=
division	/	greater than or equal to	>=
power	^	Is equal	==
modulo	%%	Is different	!=
Integer division	%/%		

Examples:

```
> 1 + 3
[1] 4
```
← 1 plus 3 equals 4

```
> 3*4+5
[1] 17
```
← 3 multiplied by 4 plus 5 is equal to 17

```
> 2^3
[1] 8
```
← 2 to the power of 3 is equal to 8

```
> 3<5
[1] TRUE
```
← 3 is less than 5

```
> 4>9
[1] FALSE
```
← 4 is greater than 9

```
> 6==13
[1] FALSE
```
← 6 is equal to 13

Assigning Objects

In R you are able to assign variables to an object. For instance, perhaps you have completed a computation in which you would like to preserve the answer. You can do this by assigning it to a letter or word. You do this by using the assignment operator <- (the less-than symbol followed by a hyphen).

For example:

```
> expenses <- 14.34
```
← We assigned $14.34 to the object '**expenses'**

```
> earnings <-26.99
```
← We assigned $26.99 to the object '**earnings'**

```
> profit <- earnings - expenses
```
← '**profit'** was assigned the value of earnings minus expenses

Tips when Assigning Objects

1. Objects are case sensitive.
   - **OFF** and **off** and **Off** are three different objects.

2. Objects must begin with a letter.

3. Objects can be more than one word. You can join words by use of a period.

```
> profits.earned <- earnings - expenses
```

4. To verify an object has been assigned correctly, type the name of the object and hit enter

```
> expenses <- 14.34
> expenses
[1] 14.34
```

← **'expenses'** are 14.34

```
> earnings <-26.99
> earnings
[1] 26.99
```

← **'earnings'** are 26.99

```
> profit <- earnings - expenses
> profit
[1] 12.65
```

← **'profit'** is the result of expenses minus earnings $12.65

5. **DO NOT** assign the letters **c, q,** and **t** as objects. They are built-in commands in R and will likely cause issues down the road.

6. If an object is already defined you will replace its value if you define it again.

'earnings' were 26.99 and are now 25.99

```
> earnings
[1] 26.99
```

```
> earnings<-25.99
> earnings
[1] 25.99
```

NOTE: If you change the value of an object any computations you run with the original DO NOT change automatically. You will need to rerun those objects.

'profit' remains 12.65 until rerun. Now **profit** is 11.65

```
> profit
[1] 12.65
```

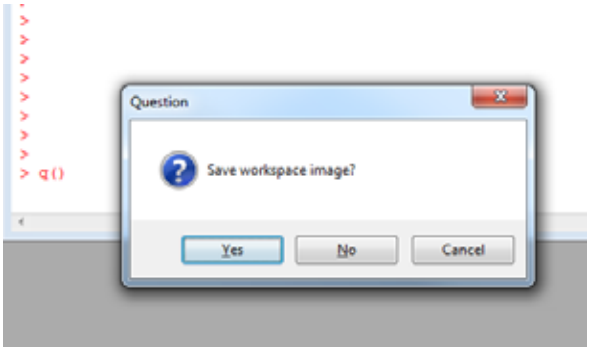
```
> profit <- earnings - expenses
> profit
[1] 11.65
```

Commands / Functions

A command (or function) is recognized as such when a set of parentheses follow it. This lets R know that it is supposed to run the function. If you do not include the parentheses following a command R will provide you with the code for that function.

For Example:

The command `> q()` prompts the Save workspace image window to open.



The command `> q` prompts the code for that function to appear.

```
> q
function (save = "default", status = 0, runLast = TRUE)
.Internal(quit(save, status, runLast))
<bytecode: 0x00000000110be158>
<environment: namespace:base>
```

Basic R functions

ls()	Lists the variables in the work space
help (name)	Provides help about the name For example **> help (ls)** returns a help guide on the function **ls()**
rm(object)	Removes the object from the work space
rm(list=ls())	Removes all objects from the work space
q()	Quit R

Vector or List

Creating a New List

Any basic objects can be a vector. A vector is an object that is a sequence or numbers or a list (as it is often referred to as). You can create a vector by using the `> c(#:#)` function.

For Example:

```
> new.list <- c(1:5)
> new.list
[1] 1 2 3 4 5

> new.list +5
[1]  6  7  8  9 10
```

The object **new.list** now contains the number 1 through 5

Here we see the results of added 5 to the values contained within **new.list**

Functions (List related)

Function	
c(#:#)	Combine values into Vector or List
length()	Will return the length of the vector (list)
sum()	Will return the sum of values in the parenthesis
max()	Will return the maximum value defined in the parenthesis
min()	Will return the minimum value defined in the parenthesis
mean()	Will return the mean of the object defined
median()	Will return the median of the object defined
seq(#, #, #)	Will generate a sequence defined by starting number, end number, and desired sequence

Examples:

```
> new.list
[1] 1 2 3 4 5
```

```
> length(new.list)
[1] 5
```

The length of **new.list** is 5 (I.e. there are 5 numbers contained within the object **new.list**)

```
> sum(new.list)
[1] 15
```

The sum of those 5 numbers in **new.list** equals 15

```
> sum(3,4,5,6,7,8,9)
[1] 42
```

The sum of the numbers 3,4,5,6,7,8,9 equals 42

```
> max(new.list)
[1] 5
```

The maximum number in **new.list** is 5

```
> max(3,1,9,4,3,24,5)
[1] 24
```

The maximum number in the sequence is 24

```
> min(new.list)
[1] 1
```

The minimum value in **new.list** is 1

```
> min(8,9,4,65,2)
[1] 2
```

The minimum value in the sequence is 2

```
> mean(new.list)
[1] 3
```

The mean of **new.list** is 3

```
> median(new.list)
[1] 3
```

The median of **new.list** is 3

```
> seq(4)
[1] 1 2 3 4
```

Creates a sequence of 1 through 4

```
> seq(2,10)
[1]  2  3  4  5  6  7  8  9 10
```

Creates a sequence of 2 through 10

```
> seq(2,10,2)
[1]  2  4  6  8 10
```

Creates a sequence of 2 through 10 by 2

```
> seq(2,5,0.5)
[1] 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Creates a sequence of 2 through 5 by 0.5

```
> new.seq <- seq(2, 12, 2)
> new.seq
[1]  2  4  6  8 10 12
```

Creates a new object called **new.seq** which contains the numbers 2,4,6,8,10,12

Strings

A string is a sequence to letters and symbols defined by quotes (either single or double). A string can contain letters or characters and contain spaces.

For Example:

```
> new.string<-'hello mom'
> new.string
[1] "hello mom"
```

```
> new.string<-'How are you today?'
> new.string
[1] "How are you today?"
```

Matrices

Matrices refer to a numeric combination of rows and columns. Matrix is often used when the all the data contained the table is the same (i.e. all numeric). One method to creating a matrix involves using `> cbind()` or `> rbind()`. The function `> cbind()` is used to combine objects into columns. The function `> rbind()` is used to combine objects into rows. Another way of creating a matrix is using the `> matrix(c(),nrow,ncol)` function. With this function you can create a matrix with a defined number of rows and columns.

Combining Row and Columns

The first step in creating a matrix is creating an object.

```
> x <- c(3,5,6,8,14)
```

Creates an object called **x** which contains the values 3, 5, 6, 8, 14

```
> y <- c(5,7,11,15,18)
```

Creates an object called **y** which contains the values 5, 7, 11, 15, 18

```
> table.by.columns <- cbind(x,y)
```

Creates a matrix called **table.by.columns** by combining **x** and **y**

```
> table.by.columns
      x  y
[1,]  3  5
[2,]  5  7
[3,]  6 11
[4,]  8 15
[5,] 14 18
```

By using the already created object **x** and **y** (see cbind()), we can create a matrix by rows.

```
> table.by.rows <- rbind(x,y)
```

Creates a matrix called **table.by.rows** by combining **x** and **y**

```
> table.by.rows
  [,1] [,2] [,3] [,4] [,5]
x    3    5    6    8   14
y    5    7   11   15   18
```

Creating a Matrix

By using the matrix() function you can create a matrix with a number of defined rows or columns.

Note: The number of rows and or columns must be divisible.

Examples:

```
> new.matrix.3x2 <- matrix(c(1,2,3,6,5,4),3)

> new.matrix.3x2
     [,1] [,2]
[1,]    1    6
[2,]    2    5
[3,]    3    4
```

Creates a matrix called **new.matrix.3x2** by combining the numbers 1, 2, 3, 6, 5, 4. The number 3 at the end defines the number of rows we want the matrix to have. We have not included column information)

```
> new.matrix.2x3 <- matrix(c(1,2,3,6,5,4),,3)

> new.matrix.2x3
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    6    4
```

Creates a matrix called **new.matrix.2x3** by combining the numbers 1, 2, 3, 6, 5, 4. The number 3 at the end defines the number of columns we want the matrix to have. Note that we left the number of rows blank

Other things you can do with a matrix:

You can:

- Extract rows, columns or numbers
- Exclude rows, columns, or numbers
- Add, subtract, multiply, divide matrices together

Dataframe

A data frame is a type of table expressed by rows and columns composing of various data types (i.e. strings and numeric). To create a data frame the `> data.frame()` function can be used.

Example:

Step 1: Create an Object

```
> Animals <- c("Cat", "Dog", "Sheep", "Beluga", "Cat", "Donkey", "Horse")
```

Step 2: Create new objects (as defined columns) based on the original Object

```
> Location <- c("Ohio", "Florida", "Florida", "California", "Montreal", "Montreal", "Ontario")

> Age <-c(3,6,1,8,4,3,7)
```

Step 3: Combine the objects to create a new data frame

```
> Bio <- data.frame(Animals, Location, Age)
> Bio
  Animals   Location Age
1     Cat       Ohio   3
2     Dog    Florida   6
3   Sheep    Florida   1
4  Beluga California   8
5     Cat   Montreal   4
6  Donkey   Montreal   3
7   Horse    Ontario   7
```

Creates a table called **Bio** by combining the objects "Animals", "Location", and "Age".

Viewing your data

As you already have seen you can view the data contained within an object by typing its name and pressing enter. If you had a very long list you can view just the first few rows of data by using the `> head()` command and you can see last few rows by using the `> tail()` command.

head()

```
> head(Bio)
  Animals   Location Age
1     Cat       Ohio   3
2     Dog    Florida   6
3   Sheep    Florida   1
4  Beluga California   8
5     Cat   Montreal   4
6  Donkey   Montreal   3
```

```
> head(Bio, 3)
  Animals Location Age
1     Cat     Ohio   3
2     Dog  Florida   6
3   Sheep  Florida   1
```

By adding a **,3** you can limit the view to the first 3 rows in the data frame.

tail()

```
> tail(Bio)
  Animals   Location Age
2     Dog    Florida   6
3   Sheep    Florida   1
4  Beluga California   8
5     Cat   Montreal   4
6  Donkey   Montreal   3
7   Horse    Ontario   7
```

```
> tail(Bio,3)
  Animals Location Age
5     Cat Montreal   4
6  Donkey Montreal   3
7   Horse  Ontario   7
```

By adding a **,3** you can limit the view to the last 3 rows in the data frame.

names()

By using the `> names()` function you can view the variable heading in the table.

```
> names(Bio)
[1] "Animals"     "Location"    "Age"
```

You can also change a variable name using the names() function.

```
> names(Bio) [1] = "Animal Type"

> names(Bio)
[1] "Animal Type" "Location"    "Age"
```

The [1] indicates that we want to change the name of the first variable in the table.

Fixing (changing) Data

You can fix your data by using the `> fix()` command. When you use this command the **Data Editor** will open and an editable table will appear that you can change, remove and add to.

Please note that when you add a String column you will not be able to view the summary of this.

```
>
>
>
>
>
>
>
>
>
>
> fix(Bio)
```

Data Editor

	Animals	Location	Age	var4
1	Cat	Ohio	3	
2	Dog	Florida	6	
3	Sheep	Florida	1	
4	Beluga	California	8	
5	Cat	Montreal	4	
6	Donkey	Montreal	3	
7	Horse	Ontario	7	

In the Data Editor you can add new variables by simply clicking and typing.

When you're done simply close the Data Editor.

Data Editor

	Animals	Location	Age	Destinati>	Cost
1	Cat	Ohio	3	Michigan	540
2	Dog	Florida	6	Kentucky	1295
3	Sheep	Florida	1	Ontario	800
4	Beluga	California	8	Florida	6527
5	Cat	Montreal	4	PEI	520
6	Donkey	Montreal	3	Ohio	750
7	Horse	Ontario	7	Ontario	2300

```
> Bio
  Animals   Location Age Destination Cost
1     Cat       Ohio   3    Michigan  540
2     Dog    Florida   6    Kentucky 1295
3   Sheep    Florida   1     Ontario  800
4  Beluga California   8     Florida 6527
5     Cat   Montreal   4         PEI  520
6  Donkey   Montreal   3        Ohio  750
7   Horse    Ontario   7     Ontario 2300
```

Getting a Summary of your Data

You can get a simple summary of your data frame by using the `> summary()` command.

```
> summary(Bio)
   Animals        Location      Age         Destination            Cost
 Beluga:1   California:1   Min.   :1.000   Length:7           Min.   : 520
 Cat   :2   Florida   :2   1st Qu.:3.000   Class :character   1st Qu.: 645
 Dog   :1   Montreal  :2   Median :4.000   Mode  :character   Median : 800
 Donkey:1   Ohio      :1   Mean   :4.571                      Mean   :1819
 Horse :1   Ontario   :1   3rd Qu.:6.500                      3rd Qu.:1798
 Sheep :1                  Max.   :8.000                      Max.   :6527
```

Please note that if you added a String column when using the fix() command you will not be able to view the summary of this. To fix this you much create a new object and add it to the data frame.

```
> Destination <- c("Michigan", "Kentucky", "Ontario", "Florida", "PEI", "Ohio", "Ontario")
> Cost <- c(540, 1295, 800, 6547, 520, 750, 2300)
> Bio <- data.frame(Animals, Location, Age, Destination, Cost)
> summary(Bio)
   Animals        Location      Age         Destination      Cost
 Beluga:1   California:1   Min.   :1.000   Florida :1    Min.   : 520
 Cat   :2   Florida   :2   1st Qu.:3.000   Kentucky:1    1st Qu.: 645
 Dog   :1   Montreal  :2   Median :4.000   Michigan:1    Median : 800
 Donkey:1   Ohio      :1   Mean   :4.571   Ohio    :1    Mean   :1822
 Horse :1   Ontario   :1   3rd Qu.:6.500   Ontario :2    3rd Qu.:1798
 Sheep :1                  Max.   :8.000   PEI     :1    Max.   :6547
```

Above we created the Destination and Cost objects and overwrote the Bio data frame to include the new columns.

To summarize individual variables within a table, use the summary() function accompanied with a $ followed by the Variable that you would like summarized.

```
> summary(Bio$Age)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   3.000   4.000   4.571   6.500   8.000
```

Bio – Name of data table
$
Age – Variable to summarize

Viewing your Data

You can limit your view of your data to rows or columns.

Rows

```
> Bio[1:4,]
  Animal Type   Location Age Destination Cost
1         Cat       Ohio   3    Michigan  540
2         Dog    Florida   6    Kentucky 1295
3       Sheep    Florida   1     Ontario  800
4      Beluga California   8     Florida 6547
```

Show rows 1 to 4 in from the Bio data table.

```
> Bio[1,]
  Animal Type Location Age Destination Cost
1         Cat     Ohio   3    Michigan  540
```

Show row 1 from the Bio data table.

Columns

Note how there are various ways to view specific variables.

```
> Bio[,4]
[1] Michigan Kentucky Ontario  Florida  PEI      Ohio     Ontario
Levels: Florida Kentucky Michigan Ohio Ontario PEI

> Bio[,"Destination"]
[1] Michigan Kentucky Ontario  Florida  PEI      Ohio     Ontario
Levels: Florida Kentucky Michigan Ohio Ontario PEI

> Bio$Destination
[1] Michigan Kentucky Ontario  Florida  PEI      Ohio     Ontario
Levels: Florida Kentucky Michigan Ohio Ontario PEI
```

In the above example, we are asking that from the data table Bio show column 4. The Levels: indicate all possible responses within that respective column.

Note how there are various ways to view specific variables.

```
> Bio[1:5,"Age"]
[1] 3 6 1 8 4
```

Show row 1 to 5 from the column variable Age

```
> Bio[1:5,3]
[1] 3 6 1 8 4
```

Show row 1 to 5 from column 3

```
> Bio$Age[1:4]
[1] 3 6 1 8
```

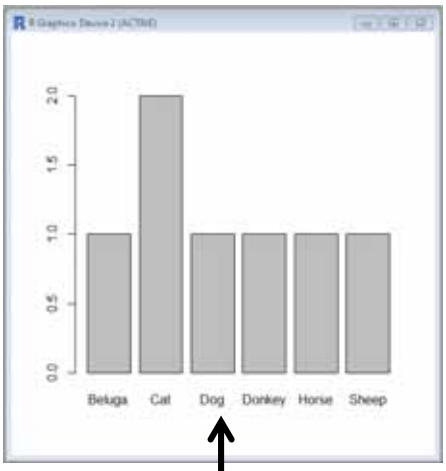
From the variable Age within the Bio data table, show rows 1 through 4

Plotting your Data

You can view your data as a chart or as a graph using the `> plot()` function. Typically the plot function will most often plot an x-y scatterplot, but if the x variable is categorical (i.e., a set of names) R will automatically plot a box-and-whisker plot.

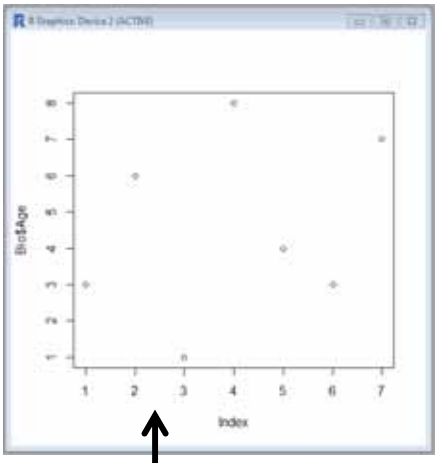
Below we have plotted the variables **Animals** and **Age** from the table **Bio**.

```
> plot(Bio$Animals)
```



Animal names plotted on the x-axis and count on the y-axis

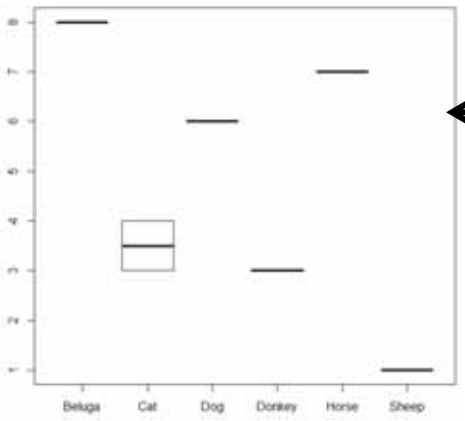
```
> plot(Bio$Age)
```



Animal age on the y-axis and animal index number on the x-axis

```
>  Bio
  Animal Type   Location Age Destination Cost
1          Cat       Ohio   3    Michigan  540
2          Dog    Florida   6    Kentucky 1295
3        Sheep    Florida   1     Ontario  800
4      Beluga California   8     Florida 6547
5          Cat   Montreal   4         PEI  520
6       Donkey   Montreal   3        Ohio  750
7        Horse    Ontario   7     Ontario 2300
```

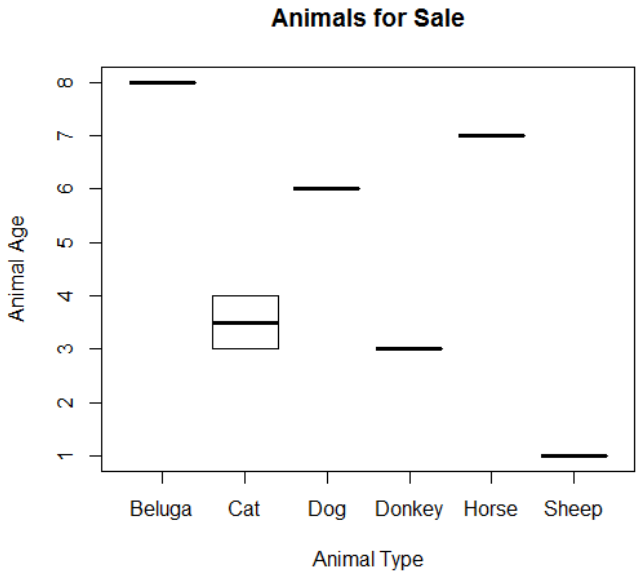
```
> plot(Bio$Animal, Bio$Age)
```



Here we have plotted Animal by Age

Notice that Cat is in a box. There are 2 cats aged 3 and 4. The centre line signifies the average cat age and the box outlines the oldest and youngest age

```
> plot(Bio$Animal, xlab="Animal Type", Bio$Age, ylab="Animal Age", main="Animals for Sale")
```



Here we have plotted Animal by Age and added x-axis and y-axis labels

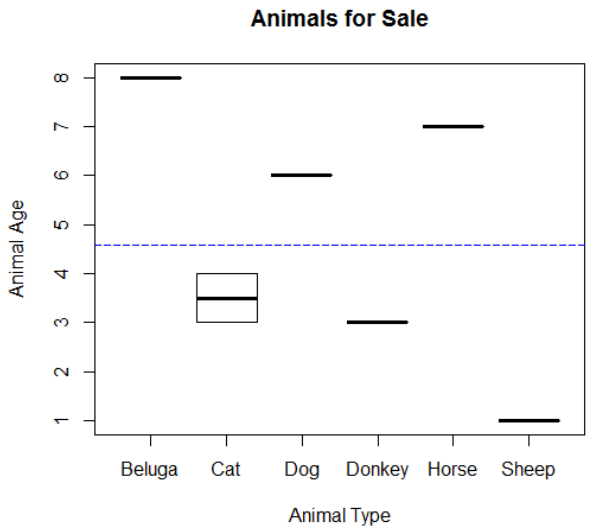
xlab – labels the x-axis
ylab – labels the y-axis
main – titles the graph

Adding Lines to a Plot

You can add lines to a graph by using the `> abline` function.

The command `> colors()` will open all 657 different colours available. Line Types available include: 1 (solid), 2 (dashed), 3 (dotted), 4 (dash-dot), 5 (larger dash), 6 (dash-smaller dash).

```
> abline(h=mean(Bio$Age), col="blue", lty="dashed")
```



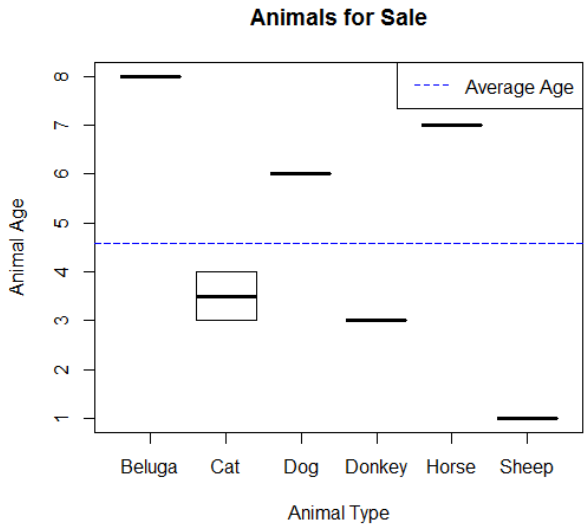
Abline – adds a line to a plot.

h – adds to y coordinate
v – adds to x coordinate
col – identifies the colour of the line
lty – identifies the line type

Adding a Legend to a Plot

You can add a legend to your graph by using the `> legend` function. Legend locations include "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and, "center".

```
> legend("topright", legend="Average Age", lty="dashed", col="blue")
```



legend – adds a legend to your graph

col – identifies the colour of the line
lty – identifies the line type

Comprehensive Function List

Basic Functions	
ls()	Lists the variables in the work space
help (name)	Provides help about the name For example **> help (ls)** returns a help guide on the function **ls()**
rm(object)	Removes the object from the work space
rm(list=ls())	Removes all objects from the work space
q()	Quit R
c(#:#)	Combine values into Vector or List
List Related Functions	
length()	Will return the length of the vector (list)
sum()	Will return the sum of values in the parenthesis
max()	Will return the maximum value defined in the parenthesis
min()	Will return the minimum value defined in the parenthesis
mean()	Will return the mean of the object defined
median()	Will return the median of the object defined
seq(#, #, #)	Will generate a sequence defined by starting number, end number, and desired sequence
Matrices	
c()	A generic function which combines its arguments
cbind()	Combines objects into columns
rbind()	Combines objects into rows
matrix(c(),nrows,ncols)	Creates a matrix with a defines number of columns and rows
Data Frames	
data.frame()	Creates a data frame with various types of data
head()	Allows you to view the first rows of data in the data frame
tail()	Allows you to view the last rows of data in the data frame
name()	Outputs the variable headings in a table
fix()	Opens the Data Editor where you can edit you data table
summary()	Outputs a summary of all the variables in the data table
Plotting	
plot()	Will most often plot an x-y scatterplot, but if the x variable is categorical (i.e., a set of names) R will automatically plot a box-and-whisker plot. xlab – labels the x-axis ylab – labels the y-axis main – Titles the graph
abline()	Adds a line to the plot h – adds to y coordinate v – adds to x coordinate col – identifies the colour of the line lty – identifies the line type