Notes
R programming

Mean

Create a vector.

```
x <- c(12,7,3,4.2,18,2,54,-21,8,-5)
```

mean(x);
[1] 8.22

Meadian

# Create the vector.

```
x <- c(12,7,3,4.2,18,2,54,-21,8,-5)
```

median(x);

[1] 5.6

MODE

The mode is the value that has highest number of occurrences in a set of data. Unike mean and median, mode can have both numeric and character data.

R does not have a standard in-built function to calculate mode. So we create a user function to calculate mode of a data set in R. This function takes the vector as input and gives the mode value as output.

# Create the function.

```
getmode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
}

# Create the vector with numbers.
v <- c(2,1,2,3,1,2,3,4,1,5,5,3,2,3)

# Calculate the mode using the user function.

result <- getmode(v)

print(result)
```

Answer: [1] 2

Addition of Two Vectors

```
c(1,2,3,4,5) + c(6,7,8,1,0)
```

```
[1]  7  9 11  5  5

 c(1,2,3,4,5) - c(6,7,8,1,0)

[1] -5 -5 -5  3  5
```

Assignment Operator

The left-pointing arrow (<–) is the assignment operator; it is composed of the

two characters `<` (less than) and `–` (dash or minus), with no intervening blanks, and is usually read as gets: “The variable `x` gets the value `c(1, 2, 3, 4)`.”

comparison operators:

== equals

`!=` not equals
`<=` less than or equals
`<` less than
`>` greater than
`>=` greater than or equals

The double-equals sign (`==`) is used for testing equality, because `=` is reserved for specifying function arguments and for assignment.

Logical values may also be used in conjunction with the logical operators:

`&` and
`|` or

Here are some simple examples:

```
> 1 == 2
[1] FALSE
> 1 != 2
[1] TRUE
```

```
 x<-c(1,2,3)
> x[1]  Gives first element of the series.
[1] 1
```

Graphing—

Run the script below, noting that a new window opens up, the graphics window, shown below.

```
# INTRO TO GRAPHING
# input a vector x with data

x <- c(2,4,4,6,6,5,5,7,3,7,3,8,9,7,9,6,4,3,4,4,6,2,2,1,2,4,6,6,8)

hist(x)
```

y <- c(1:29) Create vector y.

(input vector y of consecutive numbers from 1 through 29)

plot(x,y) **Scatter graph will be obtained**

The Help Command—

window. For example, if you know the syntax of the command you are looking

for (like mean), you would type

```
>? mean
```

and the help information will show up on screen.

```
>help.search("standard deviation")
```

To come out

Shift:

then press

w q

http://www.statmethods.net/input/datatypes.html

Data types

R - Data Types

Data Type	Example
Numeric	12.3, 5, 999
Integer	2L, 34L, 0L
Complex	3 + 2i
Character	'a' , '"good", "TRUE", '23.

Numeric

Decimal values are called numerics in R. It is the default computational data type. If we assign a decimal value to a variable x as follows, x will be of numeric type.

```
>x=10.5 # assign a decimal value
>x # print the value of x
[1] 10.5
>class(x) #print the class name of x
[1]"numeric"
```

Furthermore, even if we assign an integer to a variable k, it is still being saved as a numeric value.

```
>k = 1
>k # print the value of k
[1] 1
>class(k) #print the class name of k
[1] "numeric"
```

The fact that k is not an integer can be confirmed with the is.integer function. We will discuss how to create an integer in our next tutorial on the integer type.

```
> is.integer(k) #iskaninteger?
[1] FALSE
```

Complex

A complex value in R is defined via the pure imaginary value i.

```
>z =1+2i  #create a complex number
>z # print   the   value   of   z
[1]   1+2i
>   class(z)     #   print   the   class   name   of   z
[1]   "complex"
```

Logical

A logical value is often created via comparison between variables.

```
> x = 1; y = 2   # sample values
> z = x > y     # is x larger than y?
> z             # print the logical value
```

```
[1] FALSE
> class(z)      # print the class name of z
[1] "logical"
```

Standard logical operations are "&" (and), "|" (or), and "!" (negation).

```
> u = TRUE; v = FALSE
> u & v       # u AND v
[1] FALSE
> u | v       # u OR v
[1] TRUE
> !u          # negation of u
[1] FALSE
```

Character

A character object is used to represent string values in R. We convert objects into character values with the as.character() function:

```
> x = as.character(3.14)
> x               # print the character string
[1] "3.14"
> class(x)        # print the class name of x
[1] "character"
```

Two character values can be concatenated with the paste function.

```
> fname = "Joe"; lname ="Smith"
> paste(fname, lname)
[1] "Joe Smith"
```

Integer

We use as.integer() function to convert into integers. This converts numeric value to integer values.

```
x = 1.34
 [1] 1.34
class(x)
[1] "numeric"
y = as.integer(x)
class(y)
[1] "integer"
y
[1] 1
```

Note: to check if the value is integer or not we use is.integer() function.
In the below example ‘y’ is numerical or decimal value whereas x is integer.

```
 is.integer(y)
[1] TRUE
 is.integer(x)
[1] FALSE
```

```
# Create a vector.
```

```
apple <- c('red','green',"yellow")
print(apple)

# Get the class of the vector.
print(class(apple))
```

Lists

A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

```
# Create a list.
list1 <- list(c(2,5,3),21.3,sin)

# Print the list.
print(list1)
```

Matrices

A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.

```
# Create a matrix.
M = matrix( c('a','a','b','c','b','a'), nrow = 2, ncol = 3, byrow = TRUE)
print(M)
```

```
 B = matrix(

+ c(2, 4, 3, 1, 5, 7),

+ nrow=3,

+ ncol=2)

> B

     [,1] [,2]

[1,]    2    1

[2,]    4    5

[3,]    3    7

 > t(B) ****************************Transpose

     [,1] [,2] [,3]
```

```
[1,]    2    4    3

[2,]    1    5    7

> matrix(1:9, nrow=3, byrow=TRUE)    # fill matrix row-wise

> matrix(1:9, nrow=3)
```