Introduction to R

Adam M. Wilson

February 17, 2015

This script is available:

- SpatialAnalysisTutorials repository
- Plain text (.R) with commented text here

Variables

```
x=1
x
```

```
## [1] 1
```

We can also assign a vector to a variable:

```
x=c(5,8,14,91,3,36,14,30)
x
```

```
## [1]  5  8 14 91  3 36 14 30
```

And do simple arithmetic:

```
x+2
```

```
## [1]  7 10 16 93  5 38 16 32
```

Note that `R` is case sensitive, if you ask for `X` instead of `x`, you will get an error

```
X
```

```
Error:  object 'X' not found
```

Variable naming conventions

Naming your variables is your business, but there are 5 conventions to be aware of:

- alllowercase: *e.g.* `adjustcolor`
- period.separated: *e.g.* `plot.new`
- underscore_separated: *e.g.* `numeric_version`
- lowerCamelCase: *e.g.* `addTaskCallback`
- UpperCamelCase: *e.g.* `SignatureMethod`

Subsetting

Subset the vector using `x[ ]` notation

```
x[5]
```

```
## [1] 3
```

You can use a `:` to quickly generate a sequence:

```
1:5
```

```
## [1] 1 2 3 4 5
```

and use that to subset as well:

```
x[1:5]
```

```
## [1]  5  8 14 91  3
```

Using Functions

To calculate the mean, you could do it *manually* like this

```
(5+8+14+91+3+36+14+30)/8
```

```
## [1] 25.125
```

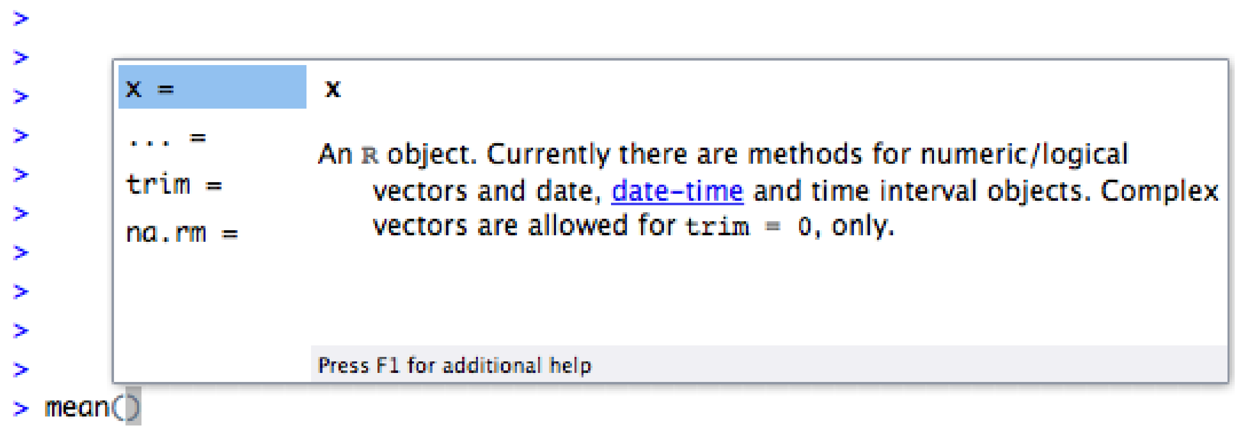
Or use a function:

```
mean(x)
```

```
## [1] 25.125
```

Type `?functionname` to learn more about a function, e.g. `?mean`. In RStudio, you can also search in the help panel. There are other arguments too: `mean(x, trim = 0, na.rm = FALSE, ...)`

In RStudio, if you press `TAB` after a function name (such as `mean()`, it will show function arguments.



```
x =
... =
trim =
na.rm =
```

x

An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for trim = 0, only.

Press F1 for additional help

```
> mean()
```

Try to calculate the mean of `c(3,6,12,89)`.

Writing functions in R is pretty easy. Let's create one to calculate the mean of vector by getting the sum and length. First think about how to break it down into parts:

```
x1= sum(x)
x2=length(x)
x1/x2
```

```
## [1] 25.125
```

Then put it all back together and create a new function called `mymean`:

```
mymean=function(f){
  sum(f)/length(f)
}

mymean(f=x)
```

```
## [1] 25.125
```

Missing data: dealing with `NA` values

But, that simple function would be vulnerable to potential problems (such as missing data).

```
x3=c(5,8,NA,91,3,NA,14,30,100)

## Calculate the mean using the new function
mymean(x3)
```

```
## [1] NA
```

```
## Use the built-in function (with and without na.rm=T)
mean(x3)
```

```
## [1] NA
```

```
mean(x3,na.rm=T)
```

```
## [1] 35.85714
```

Writing simple functions is easy, writing robust, reliable functions can be hard…

Logical values

R also has standard conditional tests to generate `TRUE` or `FALSE` values (which also behave as `0`s and `1`s. These are often useful for filtering data (e.g. identify all values greater than 5). The logical operators are `<`, `<=`, `>`, `>=`, `==` for exact equality and `!=` for inequality.

```
  x3 > 75
```

```
## [1] FALSE FALSE    NA  TRUE FALSE    NA FALSE FALSE  TRUE
```

```
  x3 == 40
```

```
## [1] FALSE FALSE    NA FALSE FALSE    NA FALSE FALSE FALSE
```

```
  x3 >   15
```

```
## [1] FALSE FALSE    NA  TRUE FALSE    NA FALSE  TRUE  TRUE
```

```
sum(x3>15,na.rm=T)
```

```
## [1] 3
```

And of course you can save the results as variables:

```
result =  x3 >  3
result
```

```
## [1]  TRUE  TRUE    NA  TRUE FALSE    NA  TRUE  TRUE  TRUE
```

> Try this: define a function that counts how many values in a vector are less than or equal to 12

Generating Data

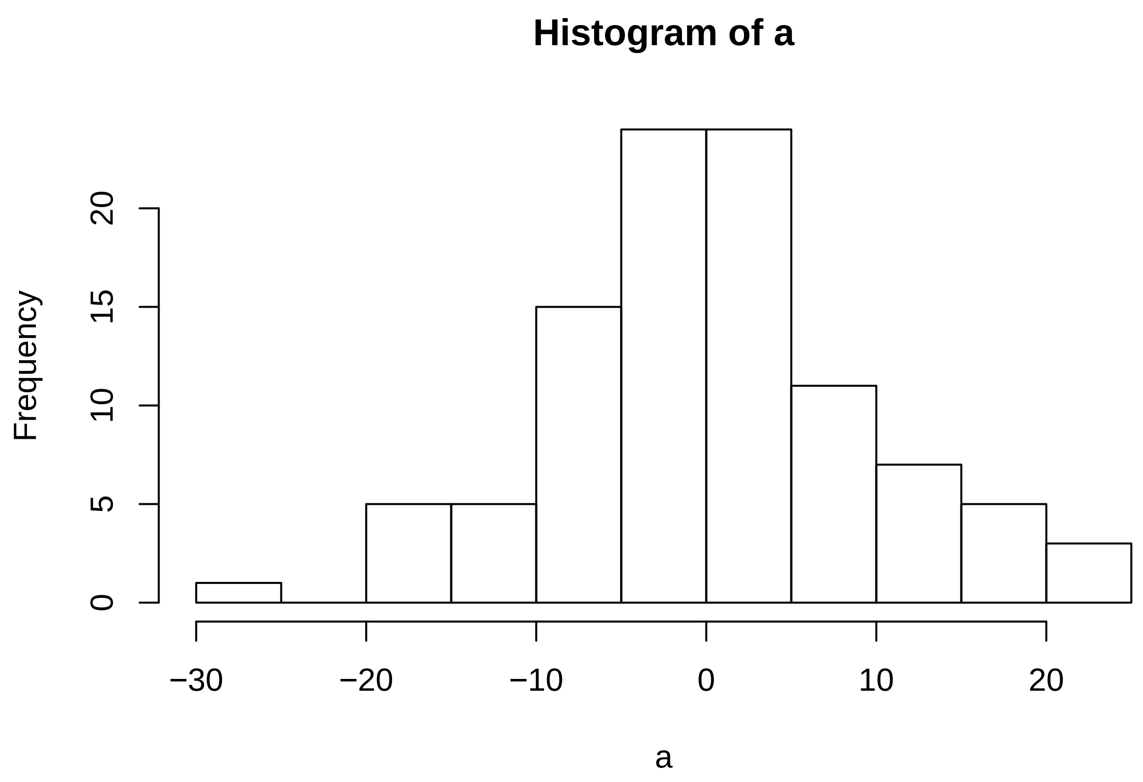
There are many ways to generate data in R such as sequences:

```
seq(from=0, to=1, by=0.25)
```

```
## [1] 0.00 0.25 0.50 0.75 1.00
```

and random numbers that follow a statistical distribution (such as the normal):

```
a=rnorm(100,mean=0,sd=10)
## illustrate with a histogram
hist(a)
```



Matrices

You can also use matrices (2-dimensional arrays of numbers):

```
y=matrix(1:30,ncol=5)
y
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    7   13   19   25
## [2,]    2    8   14   20   26
## [3,]    3    9   15   21   27
## [4,]    4   10   16   22   28
## [5,]    5   11   17   23   29
## [6,]    6   12   18   24   30
```

Which behave much like vectors:

```
y+2
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    3    9   15   21   27
## [2,]    4   10   16   22   28
## [3,]    5   11   17   23   29
## [4,]    6   12   18   24   30
## [5,]    7   13   19   25   31
## [6,]    8   14   20   26   32
```

and have 2-dimensional indexing:

```
y[2,3]
```

```
## [1] 14
```

Data Frames

Data frames are similar to matrices, but more flexible. Matrices must be all the same type (e.g. all numbers), while a data frame can include multiple data types (e.g. text, factors, numbers). Dataframes are commonly used when doing statistical modeling in R.

```
data = data.frame( x = c(11,12,14),
                   y = c(19,20,21),
                   z = c(10,9,7))
data
```

```
##    x  y  z
## 1 11 19 10
## 2 12 20  9
## 3 14 21  7
```

```
mean(data$z)
```

```
## [1] 8.666667
```

```
mean(data[["z"]])
```

```
## [1] 8.666667
```

```
mean(data[,3])
```

```
## [1] 8.666667
```

Loading Packages

One of the best things about To load a package, you can simply type `library(package)` where `package` is the name of the package you want to load. However, this only works for packages that you already have installed on your system. To install new packages, you can use `install.packages()` or use the package manager.

```
library(raster)
```

> R may ask you to choose a CRAN mirror. CRAN is the distributed network of servers that provides access to R's software. It doesn't really matter which you chose, but closer ones are likely to be faster. From RStudio, you can select the mirror under Tools→Options or just wait until it asks you.

If you don't have the packages above, install them in the package manager or by running `install.packages("raster")` (or use the package manager).