INTRODUCTION TO *R*

STAT 251

What is R

- R is a **free** software environment for statistical computing and graphics
- It runs on Windows, Mac, Linux
- R is extensible; can be expanded by installing “packages”
- **R** is command-line driven
- **RStudio** is a open source integrated development environment (IDE) with a powerful and productive user interface for R

How to get R

- R should be available on the lab computers.

What if you want R at home?

- Google it “Download R”
- **R** : https://www.r-project.org/
- **R Studio** : https://www.rstudio.com/

Help on R

There is lots of information available on the web to learn R and to use it effectively

- Try R Code School :

  tryr.codeschool.com

- Quick start for R

  http://www.statmethods.net

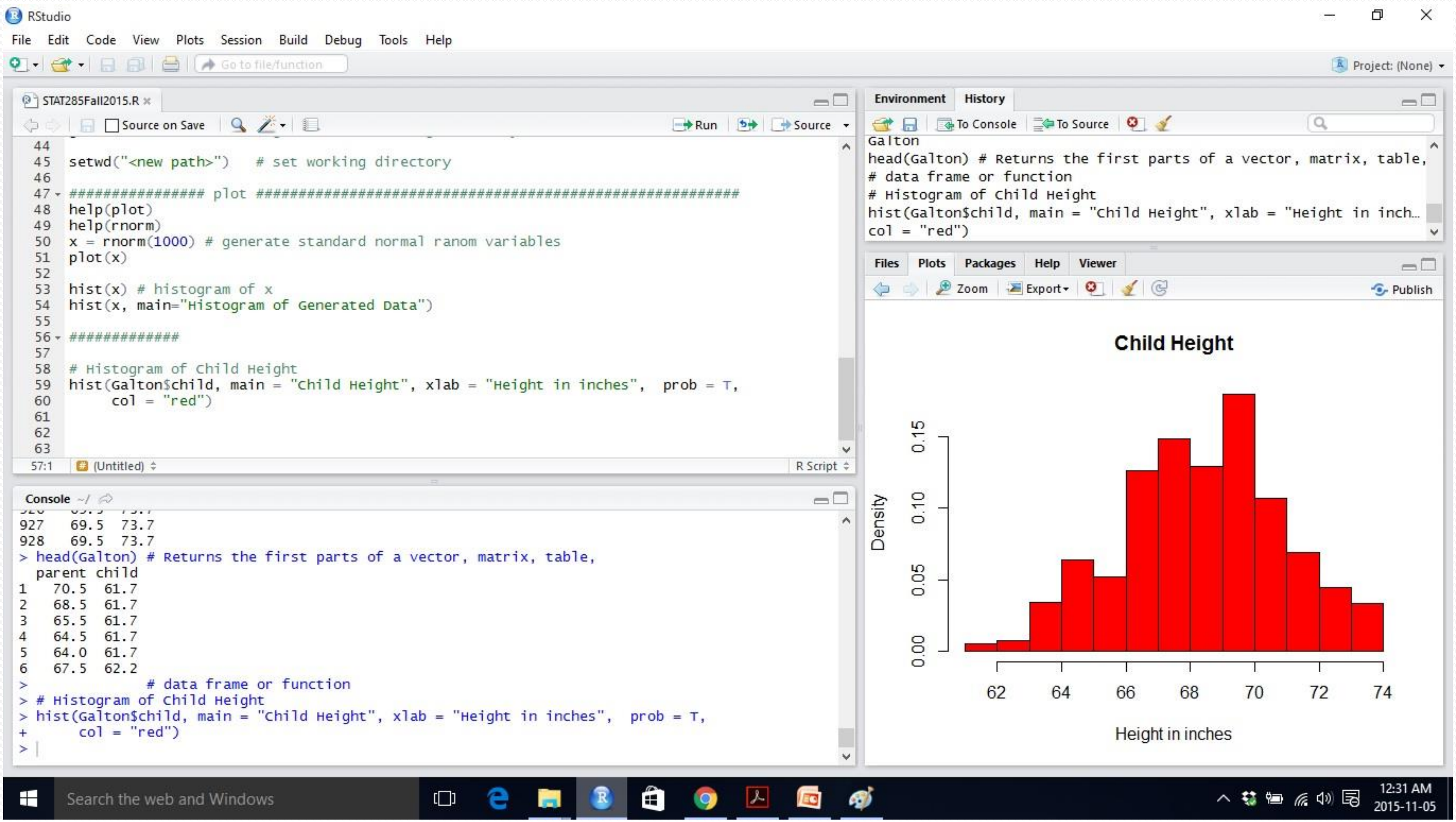
- R reference card from CRAN

RStudio

The usual R studio screen has four windows:

- Files and Data
- Console (where the action takes place)
- Workspace and History
- Files, Plots, Packages, and Help

RStudio


RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function
Project: (None)
STAT285Fall2015.R
Source on Save
Run
Source
setwd("<new path>") # set working directory
################ plot ##
help(plot)
help(rnorm)
x = rnorm(1000) # generate standard normal ranom variables
plot(x)
hist(x) # histogram of x
hist(x, main="Histogram of Generated Data")
#############
Histogram of Child Height
hist(Galton$child, main = "Child Height", xlab = "Height in inches", prob = T,
col = "red")
57:1
(Untitled)
R Script
Console ~/
927 69.5 73.7
928 69.5 73.7
> head(Galton) # Returns the first parts of a vector, matrix, table,
parent child
1 70.5 61.7
2 68.5 61.7
3 65.5 61.7
4 64.5 61.7
5 64.0 61.7
6 67.5 62.2
> # data frame or function
> # Histogram of Child Height
> hist(Galton$child, main = "Child Height", xlab = "Height in inches", prob = T,
+ col = "red")
Environment History
To Console
To Source
Galton
head(Galton) # Returns the first parts of a vector, matrix, table,
data frame or function
Histogram of Child Height
hist(Galton$child, main = "Child Height", xlab = "Height in inch…
col = "red")
Files Plots Packages Help Viewer
Zoom
Export
Publish
Child Height
Density
0.00
0.05
0.10
0.15
62
64
66
68
70
72
74
Height in inches
Search the web and Windows
12:31 AM
2015-11-05

Some Tips…

- R is case-sensitive
- R scripts are simply text files with a .R extension
- Comment your code. So it is easier to see later what you have done. comments are preceded with #
- Use up and down arrows to go through previous commands in console

Computer Lab Login Information

- **Username:** first 8 letters of your registration name

(first name, middle name, last name)

Ex. Joan Ann Beckman

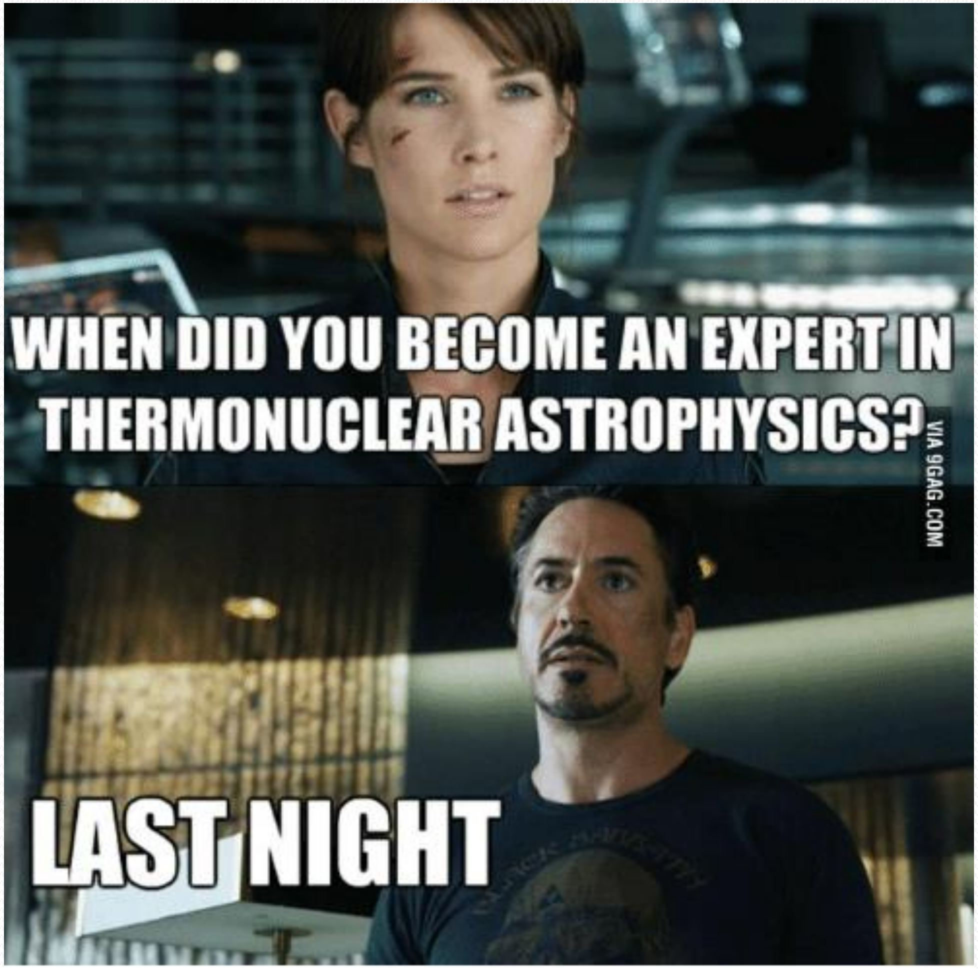
Username: joanannb

- Password: capital S followed by 1^{st} 7 digits of your student number

Ex. student number is 23567989

Username: S2356798

Crash course in R

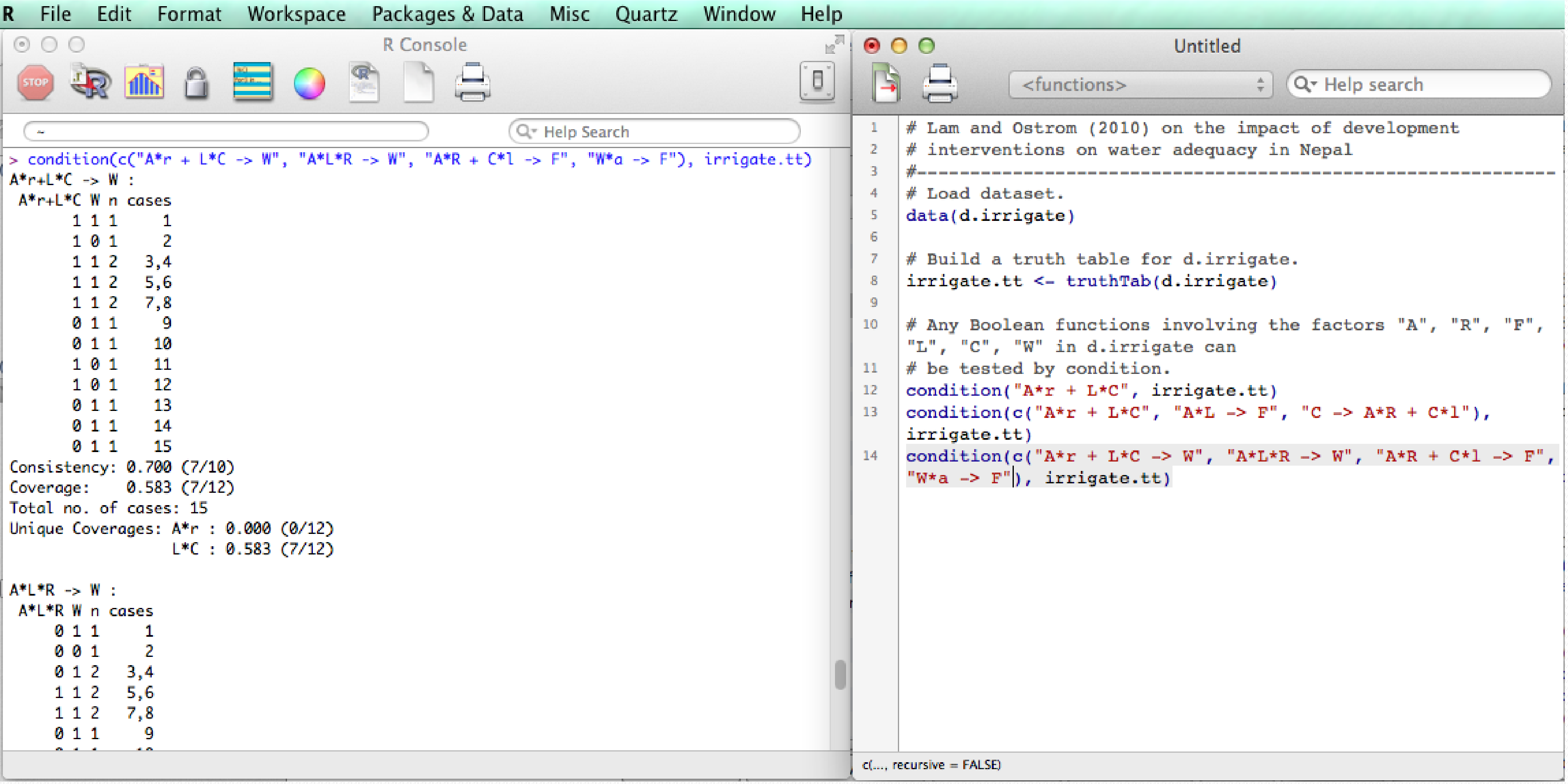


Welcome to R

- You can quit from the command line by typing q().
- You can save both the console or the source code in the document.
- Usually just save your document because you can rerun your code at any time to get the output in the console.

Welcome to R

- The console and document.



Objects

- R is an **object oriented programming language**.
- An object in R has three components: **information, a name and a class.**
- You can think of the object as a jar that contains **information**, and the **name** as the label on that jar. The **class** is the type of jar, where different types of jars store different types of **information**.

Jam and Honey Objects

Information = the goods (type of jam or honey)

Name = the label

Class = type of jar… different types of jars hold different types of jam

Names in R

- Valid names are composed of **letters**, **decimal points** and **numbers** (just not as the first character).

- ☹ **Invalid name:** 21JumpStreet <-21
- ☺ **Valid name:** JumpStreet <- 21

Names in R

- Valid names are composed of **letters**, **decimal points** and **numbers** (just not as he first character).

- General syntax is for names

name <- function(arguments)

Try these examples:

- ☹ **Invalid name:** 21JumpStreet <- 21
- ☺ **Valid name:** JumpStreet <- 21
- ☺ **Valid name:** bond **<-** 007

Example

- What is going on when we type the following?

```
x <- 7
```

Answer:

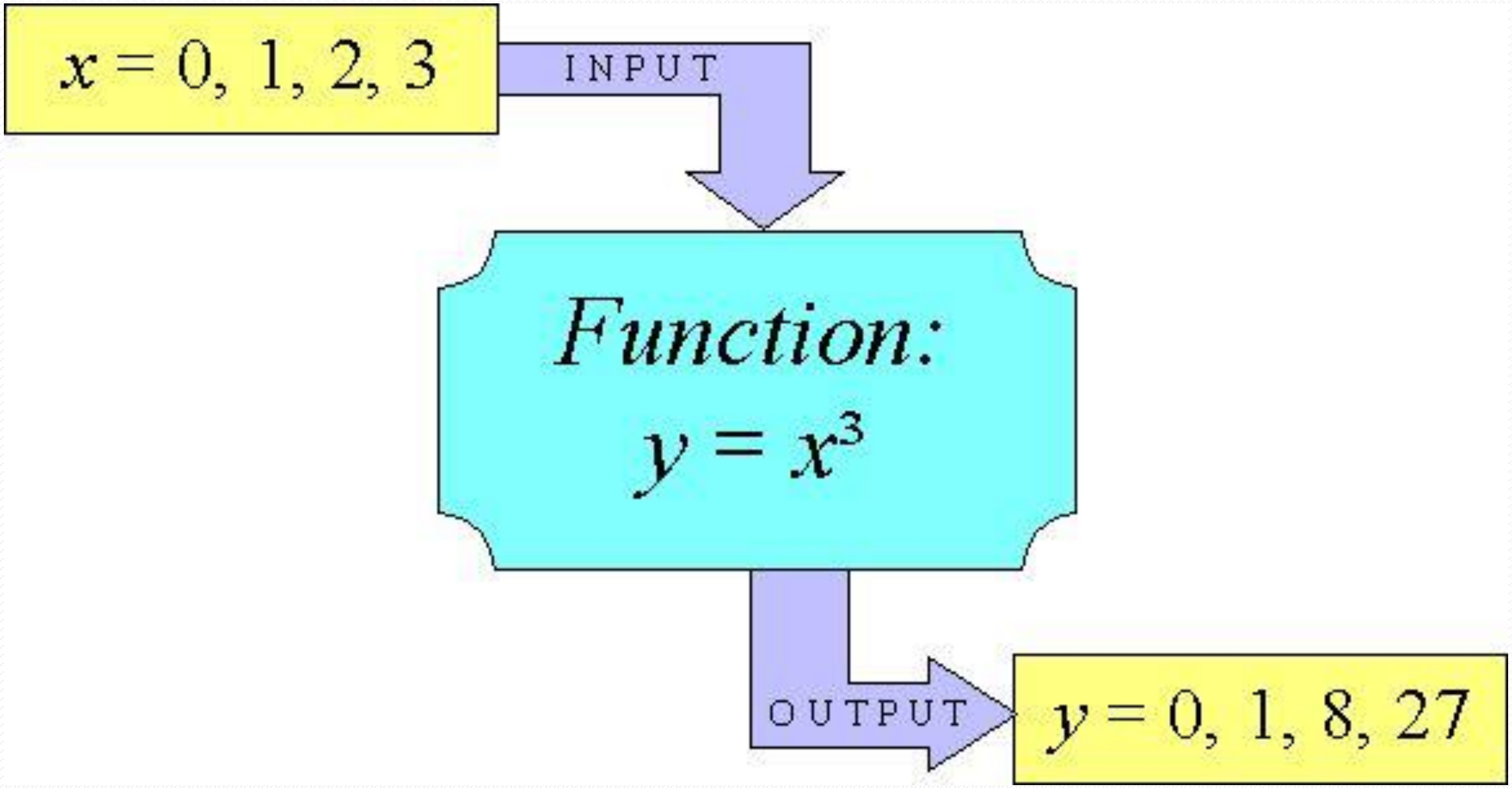
- R will create x as an object of class numerical vector.
- This vector has length 1.

Functions

- Many functions (ie. operations) that you can think of are already pre-available in R.
- Suppose you don't know what a function does? What do you do?
- Go to the R documentation
- **Try this example:**

```
?mean
```

Math Function Ex.



Mathematical operations

- Mathematical operations are simple and resemble almost every other programming language you might have already encountered. We’ll start with vectors since most mathematical operations are done on these.

- **Try this:**

myVect <- 1:14

- **Output:**

```
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14
```

- Because this is a vector, we can perform most basic mathematical functions on it like adding, subtracting, multiplying, or dividing.

myVect example

myVect

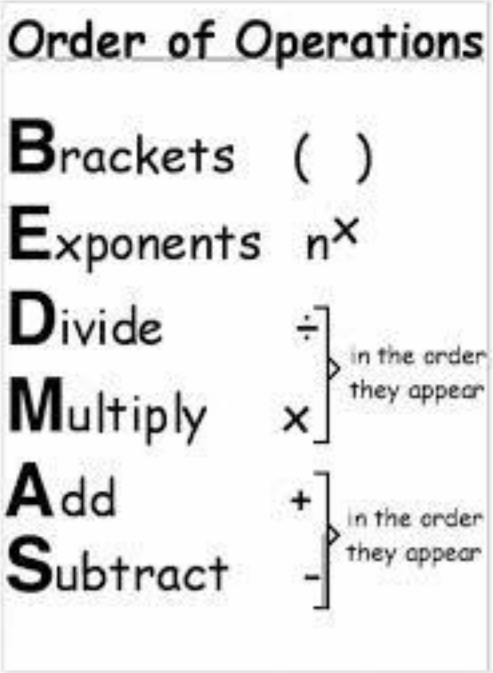
```
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14
```

• Try these ☺	Output
• myVect + 1	`[1]  2  3  4  5  6  7  8  9 10 11 12 13 14 15`
• myVect * 2	`[1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28`
• myVect * c(1, 2)	`[1]  1  4  3  8  5 12  7 16  9 20 11 24 13 28`

• myVect * c(1, 2, 3)	`[1]  1  4  9  4 10 18  7 16 27 10 22 36 13 28`
	`Warning message:`
	`In myVect * c(1, 2, 3) :`
	`  longer object length is not a multiple of shorter object length`

Don’t forget BEDMAS

- R will also obey the rules of **BEDMAS**
- That is, it performs the operations in order such that items within brackets are computed first, then exponentiation is done, then division/multiplication, and finally addition/subtraction.



More complex functions

- log(myVect) # takes the logarithm, base e

- myVect^2 # takes each element in myVect, and puts it to the power of 2

```
[1]   1   4   9  16  25  36  49  64  81 100 121 144 169 196
```

- sqrt(myVect) # Square root of each element in myVect

```
[1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427 3.000000 3.162278 3.316625 3.464102 3.605551 3.741657
```

- exp(myVect) # e^(each element in myVect)

Note: the # can be used to write comments alongside your code. R ignores your comments when it runs your code!

Common Statistical Functions in R

- **Sum of elements in a vector/matrix:** sum()
- **Average of elements in a vector/matrix:** mean()
- **Median of elements in a vector/matrix:** median()
- **Variance:** var()
- **Standard deviation:** sd()
- **Maximum value in a vector/matrix:** max()
- **Range = (min, max) of a vector/matrix:** range()
- **Summary of the values in your vector/matrix:** summary()

Example of using a stats function

- **rnorm()** generates random data from a standard Normal distribution

- **Let's try this:**

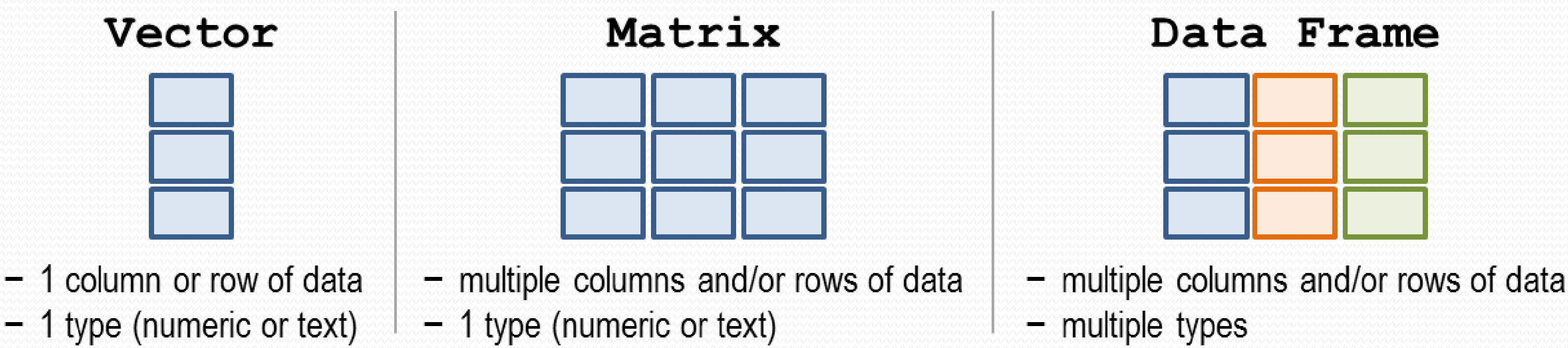
x <- rnorm(20)

summary(x) # 5 number summary of x data

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.60648 -0.75274 -0.12755 -0.07982  0.48787  2.15175
```

Three Basic Object Classes

- We'll consider the three most commonly used basic object classes: **vectors**, **matrices**, and data **frames**.



Vectors

- We already showed how to create numeric vectors of length one: x <- 7
- We can also assign vectors of ‘characters’ by writing, for example, x <- "a"
- Elements of a vector may be accessed through square brackets.

myVect   [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14

• **Try these:**	**Output:**
myVect[6]	[1] 6
myVect[c(2,14)]	[1] 2 14

Functions that are useful for vectors

- **c():** c stands for concatenate. If you write,

myVector <- c(1, 3, 7, 11), then R will store this numeric vector of length 4 in the reference named myVector.

- **rep():** create a vector of the desired length containing the same value throughout. Thus,

myVector <- rep(0, length = 5) creates a vector of length 5, where each entry is a 0.

- **Sequences:** We have two approaches for creating a sequence of numbers. 1:n or seq(1,n)
- If you look into the documentation seq() has many arguments.

• **Try this!**	**Output:**
seq(1, 10, by = 2)	[1] 1 3 5 7 9


MATRICES

Matrices

- Created using the matrix function
- The general notation is matrix(input, nrow, ncol)
- The main input when creating a matrix is a vector, plus the number of rows or columns

- **Ex.** myMatrix <- matrix(1:12, ncol = 4)

myMatrix

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

- **What is the output for this example?** matrix(0, 2, 3)

cbind() and rbind()

- We can use cbind() and rbind() to join vectors and matrices of compatible dimensions.
- **cbind():** ‘column bind’, joins your vectors / matrices column-wise (side by side).
- Unsurprisingly, **rbind():** ‘row bind’, joins your vectors / matrices row-wise (one on top of the other).

- **Try these:**

cbind(myMatrix, c(1,2,3))

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    4    7   10    1
[2,]    2    5    8   11    2
[3,]    3    6    9   12    3
```

rbind(myMatrix, c(4, 5, 6, 7))

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
[4,]    4    5    6    7
```

What is surprising?

- We have not actually modified the value of the ‘myMatrix’! If you type in myMatrix, you will find it is still the same matrix you generated at the start.

- However, if you do this:

myMatrix <- cbind(myMatrix, c(1, 2, 3))

myMatrix is changed because you are re-assigning the myMatrix reference.

Access Elements of a Matrix

- myMatrix

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

- myMatrix[2,2] # **Output:** `[1] 5`
- You can also extract entre rows or columns by leaving a specific entry in the square brackets blank.
- Ex. myMatrix[2,] means Give me all of the elements in row 2".

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

- **If you want to get the elements of column 1, what should you type?**

Data Frame

- Basically a matrix, except it allows for different columns to have different classes. ie, you can have both character and numerical columns in a data frame.

- **Let's learn by example:**

myFrame <- data.frame(num = 1:5, let = letters[1:5])

myFrame

	num	let
1	1	a
2	2	b
3	3	c
4	4	d
5	5	e

More on Names and Data Frames

- If you want to check the variable names assigned to a given data frame you can type

names(myFrame)

```
[1] "num" "let"
```

- Now, you can access a column in a data frame by using its name.
- Suppose we want to access the column of numbers. We could then write,

myFrame$num

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

Prelude to plots in R

- **Try this example:**

rand <- rnorm(20) # gives 20 random normal numbers

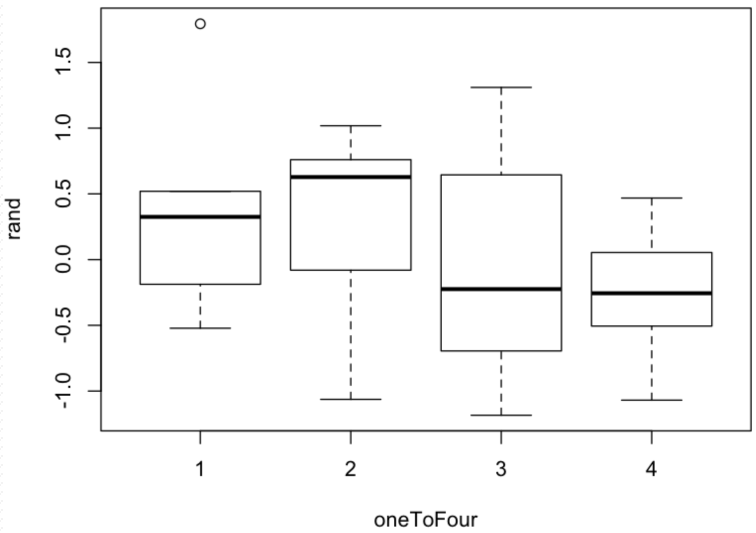
oneToFour <- rep(1:4, each = 5)

cbind(rand, oneToFour) # to display data all at once

```
             rand oneToFour
 [1,] -0.52209129         1
 [2,] -0.18761380         1
 [3,]  0.32492340         1
 [4,]  1.79380438         1
 [5,]  0.51989773         1
 [6,]  0.76031274         2
 [7,]  0.62788782         2
 [8,] -0.08062500         2
 [9,]  1.01824363         2
[10,] -1.06292914         2
[11,]  1.30991147         3
[12,] -0.22445638         3
[13,]  0.64480893         3
[14,] -0.69484508         3
[15,] -1.18388776         3
[16,]  0.46826357         4
[17,] -0.25592359         4
[18,] -1.06913791         4
[19,] -0.50583703         4
[20,]  0.05403742         4
```

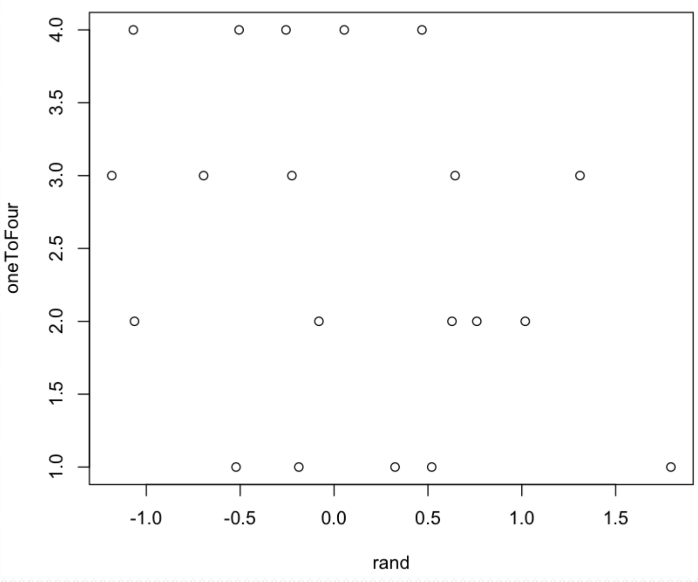
- **Try this example:**

```
rand <- rnorm(20) # gives 20 random normal numbers
oneToFour <- rep(1:4, each = 5)
cbind(rand, oneToFour) # to display data all at once
```

# Boxplot practice	**# Output**
boxplot(rand~oneToFour) # what do you think the ~ means?	

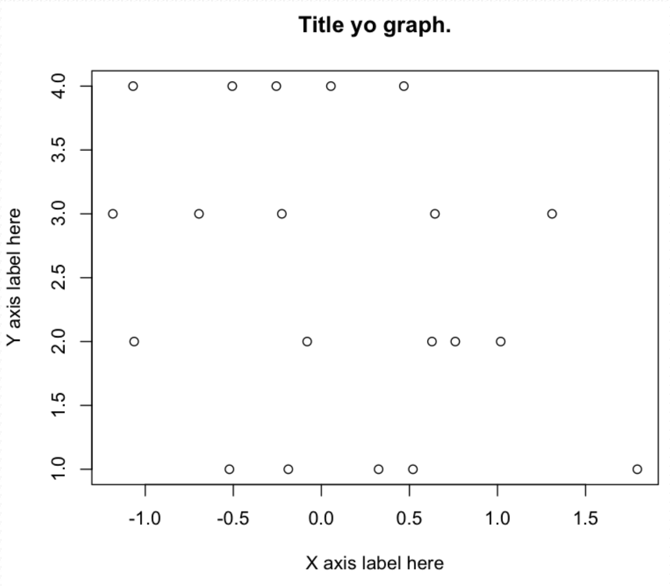
- **Try this example:**

```
rand <- rnorm(20) # gives 20 random normal numbers
oneToFour <- rep(1:4, each = 5)
cbind(rand, oneToFour) # to display data all at once
```

# Scatterplot practice	# Output
plot(oneToFour~rand) # Note the axes in the output.	

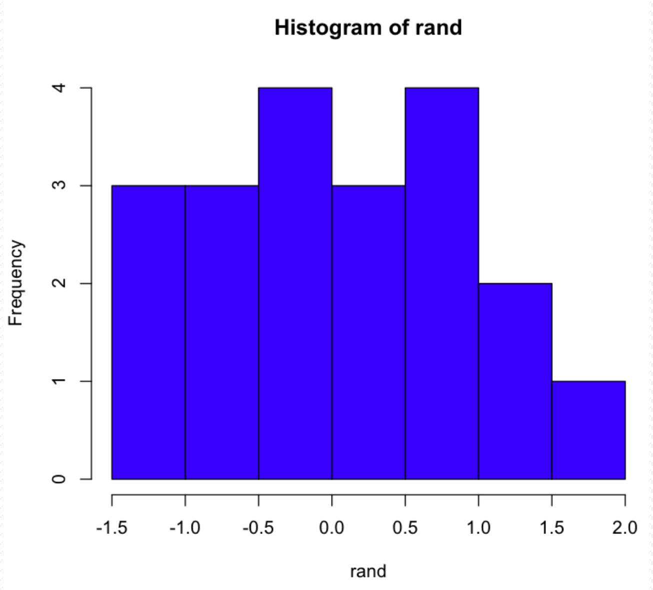
- **Try this example:**

```
rand <- rnorm(20) # gives 20 random normal numbers
oneToFour <- rep(1:4, each = 5)
cbind(rand, oneToFour) # to display data all at once
```

# Scatterplot practice	**# Output**
plot(oneToFour~rand, xlab="X axis label here", ylab="Y axis label here", main = "Title yo graph.")	

- **Try Ex. 2:**

```
rand <- rnorm(20) # gives 20 random normal numbers
oneToFour <- rep(1:4, each = 5)
cbind(rand, oneToFour) # to display data all at once
```

# Histogram practice	**# Output**
hist(rand, col="blue")	

One of many R Cheat Sheets

- **Handy R reference card:**

https://cran.r-project.org/doc/contrib/Short-refcard.pdf

FIN.