Introduction to R, Part 2

Sameh MAGDELDIN MV.Sc, 2 Ph.D

November, 2024

Al Salam Alekom,, This is a simple introduction to introduce R programming software for general use

lets start!

simple using of R {the look and feel of R}

Variable

> Variables are named memory locations reserved for storing data values.

{Variable} Variable Name

Here are a few simple guidelines for naming variables:

1. Start with a letter: The first character of a variable name must be a letter.
2. Use underscores: Underscores can be used to separate words within a variable name.
3. End with numbers: Numbers can be used at the end of a variable name.

```
varName # Example 1: characters only.
var_name # Example 2: characers and an underscore.
var_name_1 # Example 3: characters, underscores, and numbers.
```

Note that any characters following a hash symbol (#) are not interpreted by the language, making them comments within the script.

{Variable} Variable Assignment

We assign a value to a variable using either an equal sign or a less-than-dash sign.

```
varName = # Example 1: equal sign.
var_name <- # Example 2: less-than sign.
```

{Variable} Value (Data type)

The data stored in a variable is stored in memory, allowing for repeated use as long as it remains there.

```
varName = 1 # Example 1: numeric data.
var_name <- "Hello, world" # Example 2: character data.
```

{Variable} Displaying Variable Contents & Variable Usage

There are three ways to print a variable:

1. Write the variable name alone.
2. Use the built-in print function.
3. Use the built-in cat function. Note that the cat function can concatenate multiple values to be printed together.

```
varName = 1 # Example 1: numeric data.
var_name <- "Hello"
print(var_name) # Hello
```

```
## [1] "Hello"
```

```
cat(varName, "\n") # 1
```

```
## 1
```

```
cat(var_name, "Trainee", "\n") # Hello Trainee
```

```
## Hello Trainee
```

Data Types

The common data types in R are numeric, character, and logical.

Use the built-in class function to display the data type.

```
numericDataType <- 1001
characterDataType <- "1001"
logicalDataType <- TRUE
cat("The data type of the numericDataType variable is",
    class(numericDataType),
    ", and its value is",
    numericDataType,
    "\n")
```

```
## The data type of the numericDataType variable is numeric , and its value is 1001
```

```
cat("The data type of the characterDataType variable is",
    class(characterDataType),
    ", and its value is",
    characterDataType,
    "\n")
```

```
## The data type of the characterDataType variable is character , and its value is 1001
```

```
cat("The data type of the logicalDataType variable is",
    class(logicalDataType),
    ", and its value is",
    logicalDataType,
    "\n")
```

```
## The data type of the logicalDataType variable is logical , and its value is TRUE
```

What is the difference between 1001 and “1001”?

Why doesn’t the value of the numericDataType variable print when it’s called within the (“The data type of the numericDataType variable is”) character string?

Note that functions can be nested within each other.

R Objects

> R objects are structures that can hold data of specific data types or other R objects, aiming to organize and store data.

The most common R objects are:

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

{R Objects} Vector

```
?c
vector_variable <- c(1, 2, 3)
vector_variable
```

```
## [1] 1 2 3
```

```
class(vector_variable)
```

```
## [1] "numeric"
```

{R Objects} List

```
?list
list_variable <- list(vector_variable, 4, "5")
list_variable
```

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

```
class(list_variable)
```

```
## [1] "list"
```

```
# Label the elements of the list
list_variable <- list("1st"=vector_variable, "2sec"=4, "3th"="5")
list_variable
```

```
## $`1st`
## [1] 1 2 3
##
## $`2sec`
## [1] 4
##
## $`3th`
## [1] "5"
```

{R Objects} Matrix

```
?matrix
matrix_variable <- matrix(data = c(1, 2, 3, 4),
                          ncol = 2,
                          nrow = 2,
                          byrow = F,
                          dimnames = list(c("Row1", "Row2"), c("Column1", "Column2")))
matrix_variable
```

```
##      Column1 Column2
## Row1       1       3
## Row2       2       4
```

```
class(matrix_variable)
```

```
## [1] "matrix" "array"
```

{R Objects} Array

```
?array
array_variable <- array(data = c(1, 2, 3, 4),
                        dim = c(2,2,3),
                        dimnames = list(c("Row1", "Row2"), c("Column1", "Column2")))
array_variable
```

```
## , , 1
##
##      Column1 Column2
## Row1       1       3
## Row2       2       4
##
## , , 2
##
##      Column1 Column2
## Row1       1       3
## Row2       2       4
##
## , , 3
##
##      Column1 Column2
## Row1       1       3
## Row2       2       4
```

```
class(array_variable)
```

```
## [1] "array"
```

{R Objects} Factor

```
?factor
simple_vector <- c(1, 2, 3, 3, 2, 5, 6, 1, 1, 1, 7)
factor_variable <- factor(simple_vector)
factor_variable
```

```
##  [1] 1 2 3 3 2 5 6 1 1 1 7
## Levels: 1 2 3 5 6 7
```

```
class(factor_variable)
```

```
## [1] "factor"
```

```
levels(factor_variable)
```

```
## [1] "1" "2" "3" "5" "6" "7"
```

```
nlevels(factor_variable)
```

```
## [1] 6
```

{R Objects} Data Frame

```
?data.frame
vector1 <- c(1, 2, 3)
vector2 <- c("One", "Two", "Three")
data_frame_variable <- data.frame(column1 = vector1,
                                  column2 = vector2,
                                  row.names = c("row1", "row2", "row3"))
data_frame_variable
```

```
##      column1 column2
## row1       1     One
## row2       2     Two
## row3       3   Three
```

```
class(data_frame_variable)
```

```
## [1] "data.frame"
```

Accessing R Objects

> Accessing elements within R objects is commonly done by index, name, or range.

- Index: An index represents the order of an element within an object.
- Name: A name can refer to a column name, row name, or label of an element in a list, etc.
- Range: A range refers to a continuous subset of an object, such as a subset of columns, rows, or elements.

{Accessing R Objects} Access a Vector

```
another_vector <- c("A", "B", "C", "D")
another_vector[1] # single element accessed by the index of that element
```

```
## [1] "A"
```

```
another_vector[2:4] # range of elements or sub-vector accessed by a range of indices of that
elements
```

```
## [1] "B" "C" "D"
```

{Accessing R Objects} Access a List

```
another_list <- list("A", "B", "C", "D")
another_list[1]
```

```
## [[1]]
## [1] "A"
```

```
another_list[2:4]
```

```
## [[1]]
## [1] "B"
##
## [[2]]
## [1] "C"
##
## [[3]]
## [1] "D"
```

```
another_list_2 <- list("A"=1, "B"=2, "C"=3, "D"=4)
names(another_list_2)
```

```
## [1] "A" "B" "C" "D"
```

```
another_list_2["B"]
```

```
## $B
## [1] 2
```

```
another_list_2[c("B", "D")]
```

```
## $B
## [1] 2
##
## $D
## [1] 4
```

```
another_list_2$C
```

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

```
another_list_3 <- list("A", "B", "C", c("D", "E", "F"))
another_list_3[4]
```

```
## [[1]]
## [1] "D" "E" "F"
```

```
another_list_3[[4]][2]
```

```
## [1] "E"
```

{Accessing R Objects} Access a Data Frame

```
another_data_frame <- data.frame(alphabets = c("A", "B", "C", "D"),
                                 numbers = c(1, 2, 3, 4),
                                 words = c("AAA", "BBB", "CCC", "DDD"))
another_data_frame[, 1] # get the first column
```

```
## [1] "A" "B" "C" "D"
```

```
another_data_frame$alphabets # get the first column
```

```
## [1] "A" "B" "C" "D"
```

```
another_data_frame[, c(1, 3)] # get the first and third columns
```

```
##   alphabets words
## 1         A   AAA
## 2         B   BBB
## 3         C   CCC
## 4         D   DDD
```

```
colnames(another_data_frame)
```

```
## [1] "alphabets" "numbers"   "words"
```

```
another_data_frame[, c("alphabets", "words")] # get the first and third columns
```

```
##   alphabets words
## 1         A   AAA
## 2         B   BBB
## 3         C   CCC
## 4         D   DDD
```

```
another_data_frame[, -1] # exclude the first column
```

```
##   numbers words
## 1       1   AAA
## 2       2   BBB
## 3       3   CCC
## 4       4   DDD
```

```
another_data_frame[, -c(1, 3)] # exclude the first and third columns
```

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

```
another_data_frame[1:2, ] # get the first two rows
```

```
##   alphabets numbers words
## 1         A       1   AAA
## 2         B       2   BBB
```

```
rownames(another_data_frame)
```

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

```
another_data_frame["1":"2", ]
```

```
##   alphabets numbers words
## 1         A       1   AAA
## 2         B       2   BBB
```

```
another_data_frame[c("1","3"), ]
```

```
##   alphabets numbers words
## 1         A       1   AAA
## 3         C       3   CCC
```

```
another_data_frame[1:2, -1]
```

```
##   numbers words
## 1       1   AAA
## 2       2   BBB
```

Operators

> Operators are responsible for performing mathematical operations or asking questions.

{Operators} Arithmetic Operators

```
number_1 <- 10
number_2 <- 20
number_1 + number_2 # Addition
```

```
## [1] 30
```

```
number_1 - number_2 # Subtraction
```

```
## [1] -10
```

```
number_1 * number_2 # Multiplication
```

```
## [1] 200
```

```
number_1 / number_2 # Division
```

```
## [1] 0.5
```

```
number_2 / (number_1 + 0.5) # Division and addition
```

```
## [1] 1.904762
```

```
number_2 %/% (number_1 + 0.5) # Floor division and addition
```

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

```
number_1^2 # Exponentiation
```

```
## [1] 100
```

> **BEDMAS** is the order of operation you need to consider when doing math

- **B**rackets and Parentheses- 1st priority
- **E**xponents- 2nd priority
- **D**ivision- 3rd priority
- **M**ultiplication- 3rd priority
- **A**ddition- 4th priority
- **S**ubtraction- 4th priority

```
1 + 3 * 5
```

```
## [1] 16
```

```
(1 + 3) * 5
```

```
## [1] 20
```

{Operators} Asking Questions or Creating Conditions

> A collection of operators helps in making decisions by outputting a logical or Boolean data type (True or False).

{Operators} Relational Operators

```
var_1 = c(1, 2, 3, 10, 11)
var_2 = 10
var_1 == var_2 # is var_1 == var_2?
```

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

```
var_1[var_1 == var_2]
```

```
## [1] 10
```

```
var_1 != var_2 # is var_1 != var_2?
```

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

```
var_1[var_1 != var_2]
```

```
## [1]  1  2  3 11
```

```
var_1 > var_2 # is var_1 > var_2?
```

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

```
var_1[var_1 > var_2]
```

```
## [1] 11
```

```
var_1 >= var_2 # is var_1 >= var_2?
```

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

```
var_1[var_1 >= var_2]
```

```
## [1] 10 11
```

```
var_1 < var_2 # is var_1 < var_2?
```

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

```
var_1[var_1 < var_2]
```

```
## [1] 1 2 3
```

```
var_1 <= var_2 # is var_1 <= var_2?
```

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

```
var_1[var_1 <= var_2]
```

```
## [1]  1  2  3 10
```

{Operators} Logical Operators

```
var_3 <- 10
var_3 == var_2 & var_3 >= var_2
```

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

```
var_3 == var_2 & var_3 > var_2
```

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

```
var_3 == var_2 | var_3 >= var_2
```

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

```
var_3 == var_2 | var_3 > var_2
```

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

```
var_3 == var_2 & !var_3 >= var_2
```

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

```
var_3 == var_2 & !var_3 > var_2
```

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

```
!(var_3 == var_2 & !var_3 >= var_2)
```

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

```
!(var_3 == var_2 & !var_3 > var_2)
```

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

{Operators} Membership Operator

```
vector_1 <- c(1:5)
vector_1
```

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

```
value_a <- 4
value_b <- 6
value_a %in% vector_1
```

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

```
value_b %in% vector_1
```

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

Practicing R

{Practicing R} Using Some Mathematical Functions

R can do simple and complicated mathematics. it will evaluate the command and return the answer, lets try it !

```
sqrt(345.2*3/(0.7^2))
```

```
## [1] 45.97249
```

I guess R is more smarter than you expect!

R can ceil the number 3.634 to 4 –> try **ceiling(3.634)**

```
ceiling(3.634)
```

```
## [1] 4
```

R can floor the number 3.634 to 3 –> try **floor(3.634)**

```
floor(3.634)
```

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

R can round the number for you as you wish **round(x,digits=n)** ,,try only **round(x)**

```
round(5.34822343,digits=3)
```

```
## [1] 5.348
```

R can calculate also the natural log **log (x)**

or common log try **log10(x)**

```
log10(100)
```

```
## [1] 2
```

```
cos(3.141593)
```

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

even more, R knows pi !!

{Practicing R} Working With R Objects

```
cases <- c(rep("normal",5), rep("diseases", 4))
cases1 <- factor(cases)   # [as factor]
```

```
cases
```

```
## [1] "normal"   "normal"   "normal"   "normal"   "normal"   "diseases" "diseases"
## [8] "diseases" "diseases"
```

```
cases1
```

```
## [1] normal   normal   normal   normal   normal   diseases diseases diseases
## [9] diseases
## Levels: diseases normal
```

What is the difference? We tell R to store the variable as **nominal value** 1 for disease and 2 for normal (alphabetically)

also, we can use factors for **Ordinal variables**

```
rating <- c(rep("a",3),rep("b",7),rep("c",5))
rating1 <- ordered(rating) # rank 1=a, 2=b, 3=c
```

```
rating1
```

```
##  [1] a a a b b b b b b b c c c c c
## Levels: a < b < c
```

```
x1 <- c(1,2,3,4,5)
x2 <- c("ali", "ahmed","mohamed","amr", "khaled")
data <- data.frame(x1,x2)   # [as data frame;different column have different mode]
```

```
data
```

```
##   x1      x2
## 1  1     ali
## 2  2   ahmed
## 3  3 mohamed
## 4  4     amr
## 5  5  khaled
```

```
x3 <- c("a","b","a","a","b")
```

```
data1 <- data.frame(x1,x2,x3)
```

```
data1
```

```
##   x1      x2 x3
## 1  1     ali  a
## 2  2   ahmed  b
## 3  3 mohamed  a
## 4  4     amr  a
## 5  5  khaled  b
```

```
z  <-  matrix(rnorm(30,5,.5),nrow=5,ncol=5) #generate matrix of 30 value[normal values] in 5 rows and 6 columns
```

```
## Warning in matrix(rnorm(30, 5, 0.5), nrow = 5, ncol = 5): data length differs
## from size of matrix: [30 != 5 x 5]
```

```
z
```

```
##          [,1]     [,2]     [,3]     [,4]     [,5]
## [1,] 5.094870 4.864446 4.477096 4.688240 4.241666
## [2,] 5.065842 5.394683 4.857267 4.716948 3.965121
## [3,] 4.576897 4.412491 4.399020 4.375680 5.024641
## [4,] 4.774524 5.046562 4.875029 5.002275 4.517581
## [5,] 5.338626 5.168287 4.820218 4.715600 4.490772
```

note that in matrix, it must have the same mode(numeric, character, etc.)

Str() [structure of an object] is an important function to show what kind of data the variable are stroed in

```
str(rating)
```

```
##  chr [1:15] "a" "a" "a" "b" "b" "b" "b" "b" "b" "b" "c" "c" "c" "c" "c"
```

```
str(data1)
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ x1: num  1 2 3 4 5
##  $ x2: chr  "ali" "ahmed" "mohamed" "amr" ...
##  $ x3: chr  "a" "b" "a" "a" ...
```

```
str(z)
```

```
##  num [1:5, 1:5] 5.09 5.07 4.58 4.77 5.34 ...
```

alternatively, you can use **class()** function

```
class(rating)
```

```
## [1] "character"
```

```
class(data1)
```

```
## [1] "data.frame"
```

```
class(z)
```

```
## [1] "matrix" "array"
```

also, you can get the dimension of the variable using **dim()** function

try it!

```
dim(data)
```

```
## [1] 5 2
```

```
dim(z)
```

```
## [1] 5 5
```

as we talked earlier, Matrices are 2-dimensional array

Lets generate some random matrices

```
m1 <- matrix(5,4,7)# make a matrix of number 5 in 4 rows and 7 columns
```

```
m1
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]    5    5    5    5    5    5    5
## [2,]    5    5    5    5    5    5    5
## [3,]    5    5    5    5    5    5    5
## [4,]    5    5    5    5    5    5    5
```

```
m2 <- matrix(1:10, ncol = 2)  # make a matrix from 1 to 10 in 2 columns
```

```
m2
```

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

```
m3  <-  matrix(rnorm(30),nrow=5,ncol=6) # make a matrix of 30 values (normal random ), in 5 r
ows and 6 columns
```

```
m3
```

```
##            [,1]       [,2]       [,3]       [,4]      [,5]        [,6]
## [1,] -2.3208330  1.3385394 -0.6701104  0.9506944  0.890293 -0.15150275
## [2,] -0.1463549 -0.5316818  0.2111467  0.8407450  1.196536 -0.80048911
## [3,]  1.4845319  0.3141349 -0.7528566 -0.2830465 -1.585122 -0.23379879
## [4,]  0.5574424  0.2941239  0.5517349 -0.2496218 -1.150552 -0.04905099
## [5,] -1.7275210  0.5618890 -0.6834906 -0.4283194 -0.430534 -0.52051995
```

```
m4<- matrix(sample(15,90,T),9)  # choose number up to 15, select 90 random,T "probability", a
nd 9 rows
```

```
m4
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]   13    9    2   14    6    2    6    4   12    15
##  [2,]    4   11    9   11   12   12   12    1    3    14
##  [3,]    4    8    2    5    1   11    4   12    9     4
##  [4,]    3    8    7    2    6   11   15   12    5     5
##  [5,]   15    1   13   12   13    8   14    3    8    12
##  [6,]    5   13    2    5   10    4    3    9    1    13
##  [7,]    1    3   12   13   13   15   12    4    5     2
##  [8,]    3    3    1   11    9    9    1   13   15    13
##  [9,]    3   15   10    5    7    3    7    4   14     3
```

There are plenty to say about data frame because they are the primary data structure in R

as we said, data frame are 2 dimensional array in which each column contains measurement of one variable. here columns might be different in entery (factor, numeric…etc)

lets retrieve the data stored on your PC. R contains several built in examples

```
head(mtcars) # just show the first few hits (6 as default) of the data named mtcars. you can
also say
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

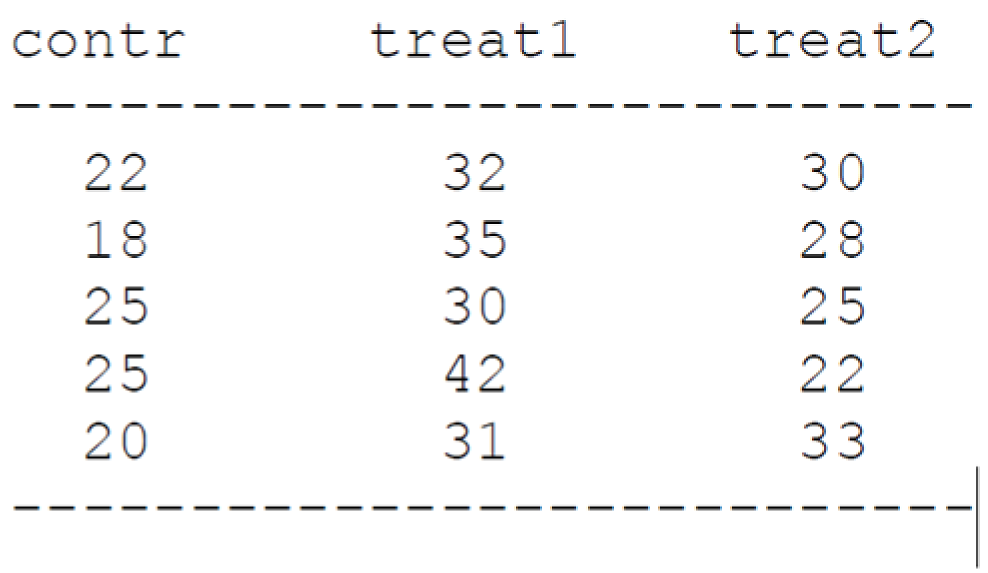
```
head(mtcars,3) # this will show the first 3 hits or,,,
```

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710    22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
```

```
tail(mtcars)
```

```
##                 mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

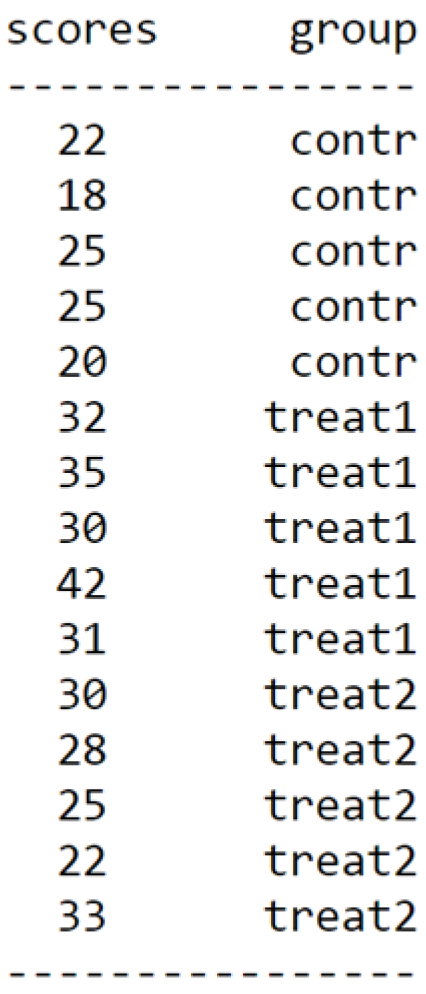
so, what do you think about this table?



contr	treat1	treat2
22	32	30
18	35	28
25	30	25
25	42	22
20	31	33

my graph

In fact, it is not a data frame, because the reading has been divided into 3 parts, a correct data frame should have a name of the variable at one column and the value in another column like this



scores	group
22	contr
18	contr
25	contr
25	contr
20	contr
32	treat1
35	treat1
30	treat1
42	treat1
31	treat1
30	treat2
28	treat2
25	treat2
22	treat2
33	treat2

my graph

OK back to mtcars data

```
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
mtcars[4,7]  # will return the value in row 4 and column 7 "19.44"
```

```
## [1] 19.44
```

```
mtcars[1:3,] # will cal the first 3 rows
```

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710    22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
```

```
mtcars[,3] # will return all values in column 3
```

```
##  [1] 160.0 160.0 108.0 258.0 360.0 225.0 360.0 146.7 140.8 167.6 167.6 275.8
## [13] 275.8 275.8 472.0 460.0 440.0  78.7  75.7  71.1 120.1 318.0 304.0 350.0
## [25] 400.0  79.0 120.3  95.1 351.0 145.0 301.0 121.0
```

```
 mtcars[c(1,3,7,13),] # will return rows 1,3,7,13 all columns
```

```
##             mpg cyl  disp  hp drat   wt  qsec vs am gear carb
## Mazda RX4  21.0   6 160.0 110 3.90 2.62 16.46  0  1    4    4
## Datsun 710 22.8   4 108.0  93 3.85 2.32 18.61  1  1    4    1
## Duster 360 14.3   8 360.0 245 3.21 3.57 15.84  0  0    3    4
## Merc 450SL 17.3   8 275.8 180 3.07 3.73 17.60  0  0    3    3
```

```
mtcars[c(1,3,7,13),1] # will column 1 only for the rows 1,3,7,and 13
```

```
## [1] 21.0 22.8 14.3 17.3
```

to summarize your data frame use **summary()** function

```
summary(mtcars)
```

```
##       mpg             cyl             disp             hp       
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0  
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
##       drat             wt             qsec             vs        
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
##        am              gear            carb      
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
##  Median :0.0000   Median :4.000   Median :2.000  
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812  
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000  
```

$ (dollar sign refers to the column inside the data frame)

```
mtcars$carb
```

```
##  [1] 4 4 1 1 2 1 4 2 2 4 4 3 3 3 4 4 4 1 2 1 1 2 2 4 2 1 2 2 4 6 8 2
```

it is very easy to do some process within the data frame for example to find if there is a correlation between 2 columns in the data frame

```
?cor
cor(mtcars$carb,mtcars$gear) # by default its pearson correlation
```

```
## [1] 0.2740728
```

```
cor(mtcars$cyl,mtcars$disp)
```

```
## [1] 0.9020329
```

OK, so far so good !

In general, you need to use the function **ls()** to list the stored variable in your work space

try ls()

{Practicing R} Remembering Some Operators

R can tell you if the expression you entered is correct or not (true or false). He will return his opinion to you

lets try it

```
5+5 == 11    #(use double equals)
```

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

```
2*2 < 2*10
```

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

```
5 == 5 & 10 < 20
```

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

```
10 < 20 & 35 < 10
```

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

```
10 < 20 | 35 < 10
```

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

```
10 > 20 | 20 > 30
```

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

Decision Making

> Decision-making structures allow you to perform different actions based on the answer to a specific question.

{Decision Making} if statement

The logical values (Boolean data type (TRUE, FALSE)) can be used in several ways, most commonly in the if statement.

```
a <- 33
b <- 200
the_answer <- b > a
the_answer
```

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

```
if (the_answer) {
  print("b is greater than a")
}
```

```
## [1] "b is greater than a"
```

{Decision Making} if-else and if-else-if-else

{Decision Making} if-else

> If the answer to the question is true, perform the action in the if block. Otherwise, if the answer is false, perform the action in the else block.

```
a <- 33
b <- 33

if (b == a & a > b) {
  print("b is greater than a")
} else {
  print("Something is wrong")
}
```

```
## [1] "Something is wrong"
```

```
a <- 33
b <- 33

if (b == a | a > b) {
  print("b is greater than a")
} else {
  print("Something is wrong")
}
```

```
## [1] "b is greater than a"
```

{Decision Making} if-else-if-else

The else-if keyword is R’s way of saying “if the previous conditions were not true, then try this condition”

```
a <- 33
b <- 33

if (b > a) {
  print("b is greater than a")
 if (a == b) {
  print ("a and b are equal")
 }
}
# The code will not print anything because the first if condition is false, and the second if
condition is nested within the first.
```

```
a <- 33
b <- 33

if (b > a) {
  print("b is greater than a")
  } else if (a == b) {
  print ("a and b are equal")
}
```

```
## [1] "a and b are equal"
```

```
a <- 33
b <- 33

if (b == a) {
  print("a and b are equal")
  } else if (a > b) {
  print("b is greater than a")
}
```

```
## [1] "a and b are equal"
```

```
a <- 33
b <- 33

if (b != a) {
  print("a and b are not equal")
  } else if (a > b) {
  print("b is greater than a")
  } else {
  print("a and b are equal")
}
```

```
## [1] "a and b are equal"
```

Loops

> Loops are used to repeat an action a specific number of times. The number of repetitions can be specified by a number or a condition that eventually becomes false.

{Loops} for loop

> A for loop repeats actions a specified number of times, limited to the length of an object’s elements.

```
num <- c(1, 2, 3, 4, 5, 6)
cat("The length of the vector is", length(num), "\n")
```

```
## The length of the vector is 6
```

```
for (i in num) {
  cat("The i variable contains", i, "\n")
}
```

```
## The i variable contains 1
## The i variable contains 2
## The i variable contains 3
## The i variable contains 4
## The i variable contains 5
## The i variable contains 6
```

```
num <- c(1, 2, 3, 4, 5, 6)

for (i in num) {
  print(i + 10)
}
```

```
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
```

```
num <- c(1, 2, 3, 4, 5, 6)

for (i in num) {
  print(i)
  if (i + 10 == 15){
    print("It is 50!!")
    break
  }
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] "It is 50!!"
```

Functions

> There are many built-in functions in R, and user-defined functions can also be created.

R can do a lot of functions.

R can generate a sequence

```
seq(1:12)  #I'm asking R to generate a sequence from 1 to 12
```

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

```
seq(1,12,2) # note that 1st,2nd, 3rd number are first, last, increments
```

```
## [1]  1  3  5  7  9 11
```

R can generate a randomized number

```
rnorm(10,mean=2)  #rnorm(n, mean = 0, sd = 1)
```

```
##  [1] 1.5004414 1.3449721 1.1007264 2.1822929 2.5218628 1.5519061 1.7311136
##  [8] 0.1743066 0.5794554 1.1519215
```

In any case you are in trouble, type **help**(function name) to see how the function works

try **help(rnorm)**

In R you can easily create your own function. Lets try it

```
myfunction <- function (x) x+2*3
myfunction(2)  # Guess the result?
```

```
## [1] 8
```

```
# **BEDMAS** is the order of operation you need to consider when doing math

#B rackets and Parentheses- 1st priority
#E xponents- 2nd priority
#D ivision- 3rd priority
#M ultiplication- 3rd priority
#A ddition- 4th priority
#S ubtraction- 4th priority
```

or little bit more complex formula of 2 variables

```
f <- function(x,y) {c(x+1, y+4)}
f(1,3)
```

```
## [1] 2 7
```

or you can also define some variables within your function

```
f2 <- function(x,y=3) {c(x+1, y+(4*x))}
f2(2)
```

```
## [1]  3 11
```

Dealing with Missing Values (NA)

In R, missing values are termed NA while impossible values returned as NaN

lets test a simple data

```
missing <- c(3,4,6,76,NA,54,NA)
```

to figure out if your data have NA (blank cells in excel with no data)

use **is.na() or summary()** function

lets try it

```
is.na(missing)
```

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

```
summary(missing)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
##     3.0     4.0     6.0    28.6    54.0    76.0       2
```

now, lets construct a data frame with missing data

```
a1 <- c(1,2,3,4,5)
a2 <- c(23,NA,52,1,NA)
data3 <- data.frame(a1,a2)
```

```
data3
```

```
##   a1 a2
## 1  1 23
## 2  2 NA
## 3  3 52
## 4  4  1
## 5  5 NA
```

you can exclude the missing data from your set using **na.omit()** function

```
md <- na.omit(data3)
```

```
md
```

```
##   a1 a2
## 1  1 23
## 3  3 52
## 4  4  1
```

dealing with NA is very important in R. it is important to know if your data contains Na or not and what do you want to do with these hits

lets see a simple example

```
a1
```

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

```
a2
```

```
## [1] 23 NA 52  1 NA
```

```
mean (a1)
```

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

```
mean(a2)
```

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

```
sum(a2)
```

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

Here, R did not calculate the mean of a2 simply because it contains Na

you need to tell R here to exclude these hits and calculate the mean of remaining values

to do that,,try

```
mean(a2, na.rm=T) # mean(a2, na.rm=TRUE)
```

```
## [1] 25.33333
```

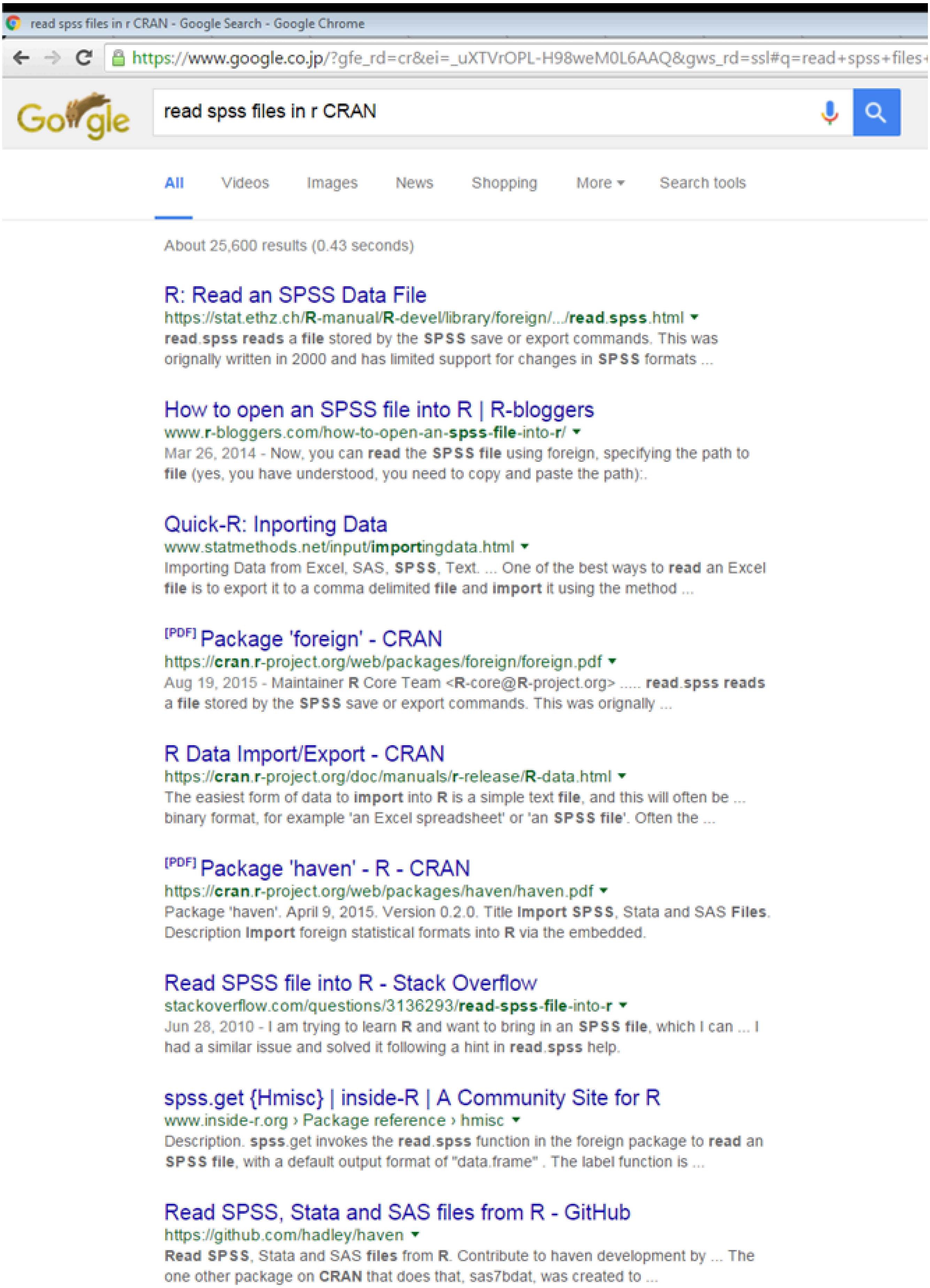
```
sum(a2, na.rm=T)
```

```
## [1] 76
```

Finding Appropriate Functionality

> As of 2024, there are over 19,000 R packages available on CRAN. So the question is, how do i search for the package i need?

Well, the first thing to do is to search on the Google


read spss files in r CRAN - Google Search - Google Chrome
https://www.google.co.jp/?gfe_rd=cr&ei=_uXTVrOPL-H98weM0L6AAQ&gws_rd=ssl#q=read+spss+files
read spss files in r CRAN
All
Videos
Images
News
Shopping
More
Search tools

About 25,600 results (0.43 seconds)

R: Read an SPSS Data File
https://stat.ethz.ch/R-manual/R-devel/library/foreign/.../read.spss.html
read.spss reads a file stored by the SPSS save or export commands. This was orignally written in 2000 and has limited support for changes in SPSS formats ...

How to open an SPSS file into R | R-bloggers
www.r-bloggers.com/how-to-open-an-spss-file-into-r/
Mar 26, 2014 - Now, you can read the SPSS file using foreign, specifying the path to file (yes, you have understood, you need to copy and paste the path):.

Quick-R: Inporting Data
www.statmethods.net/input/importingdata.html
Importing Data from Excel, SAS, SPSS, Text. ... One of the best ways to read an Excel file is to export it to a comma delimited file and import it using the method ...

[PDF] Package 'foreign' - CRAN
https://cran.r-project.org/web/packages/foreign/foreign.pdf
Aug 19, 2015 - Maintainer R Core Team <R-core@R-project.org> ..... read.spss reads a file stored by the SPSS save or export commands. This was orignally ...

R Data Import/Export - CRAN
https://cran.r-project.org/doc/manuals/r-release/R-data.html
The easiest form of data to import into R is a simple text file, and this will often be ... binary format, for example 'an Excel spreadsheet' or 'an SPSS file'. Often the ...

[PDF] Package 'haven' - R - CRAN
https://cran.r-project.org/web/packages/haven/haven.pdf
Package 'haven'. April 9, 2015. Version 0.2.0. Title Import SPSS, Stata and SAS Files. Description Import foreign statistical formats into R via the embedded.

Read SPSS file into R - Stack Overflow
stackoverflow.com/questions/3136293/read-spss-file-into-r
Jun 28, 2010 - I am trying to learn R and want to bring in an SPSS file, which I can ... I had a similar issue and solved it following a hint in read.spss help.

spss.get {Hmisc} | inside-R | A Community Site for R
www.inside-r.org › Package reference › hmisc
Description. spss.get invokes the read.spss function in the foreign package to read an SPSS file, with a default output format of "data.frame" . The label function is ...

Read SPSS, Stata and SAS files from R - GitHub
https://github.com/hadley/haven
Read SPSS, Stata and SAS files from R. Contribute to haven development by ... The one other package on CRAN that does that, sas7bdat, was created to ...

my graph

Search for the most frequently updated packages

Read pdf manual, there is always examples to replicate

Search for Vignettes, which are tutorials

Then its your choice

Finding Your Mistake

> When starting your first codes with R, you are more liable to get several error messages from R. Don’t be frustrated, check points to consider is:

1. **Is your data properly loaded? [can u see it in the environment]**
2. **Is your package installed and loaded [some times not installed, or not loaded]**
3. **your data contains NA.?**
4. **Your code spelling?**
5. **Try cutting aesthetic part from the code and start with basic code first.**
6. **After all, paste the error message to Google?**
7. **paste the error message to stack over flow?**

the MORE
YOU PRACTICE
THE BETTER
YOU GET
©2014 KindergartenWorks

my graph