Introduction to R (+ Python)

Programming in Psychological Science

Michael Nunez & Hannes Rosenbusch

04-01-2022

Meet the team

- R/Python teachers:
  - Michael Nunez (m.d.nunez@uva.nl)
  - Hannes Rosenbusch (h.rosenbusch@uva.nl)
- Teaching assistants:
  - Enrico Erler
  - Leonhard Volz
  - Clara Sophie Vetter
  - Gergely Bence
- Write stuff on Slack / Canvas!

Our goal

Make you quit psychology!

Course structure

Mondays: 13-16 Lecture + Practical + Q&A

Wednesdays: 13-16 Lecture + Practical + Q&A

Fridays: 13-16 Practical + Q&A

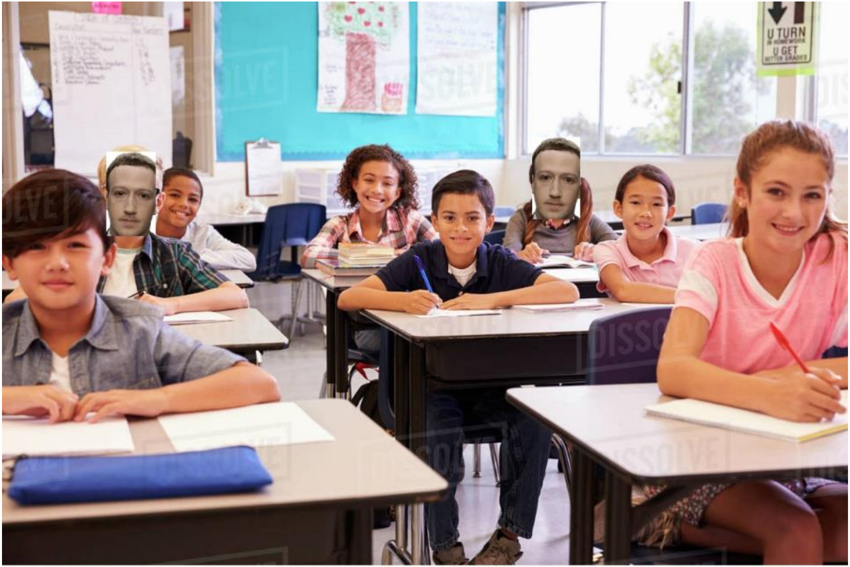
Assignments

- Coding puzzles
- Uploaded to Canvas on Mondays
- Deadline Fridays (last one Wednesday the week after)
- 60% of grade

Exam

- Last Friday
- On your laptops
- Open book
- Coding questions
- 40% of the grade

Who are the impostors?



Pep talk

- This is an intense course!
- If you decide to commit, it will be the most rewarding
- When failing - **ask questions**, work together, use Google, Stackexchange, Don’t cheat!
- Have fun in the process!

The Python parts

- R lecture is the main lecture
- For each R lecture there is a Python video
- Python questions (10% of grade)
- Assignments include Python exercises

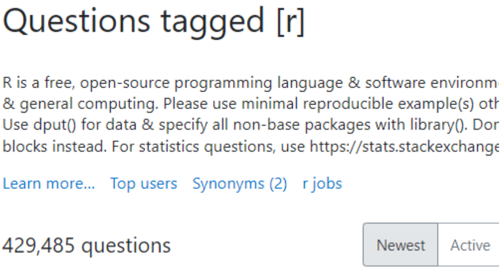
What Is R and Why Should You Learn It?

What is R?

R is great for DATA:

- Data (pre-)processing
- Data visualisation
- Data mining/machine learning
- Statistical analysis
- Programming in the wider sense

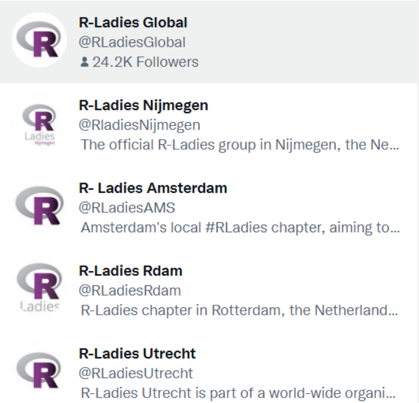
It has a huge community



Questions tagged [r]

R is a free, open-source programming language & software environm
& general computing. Please use minimal reproducible example(s) oth
Use dput() for data & specify all non-base packages with library(). Dor
blocks instead. For statistics questions, use https://stats.stackexchange

429,485 questions



R-Ladies Global
@RLadiesGlobal
24.2K Followers

R-Ladies Nijmegen
@RladiesNijmegen
The official R-Ladies group in Nijmegen, the Ne...

R- Ladies Amsterdam
@RLadiesAMS
Amsterdam's local #RLadies chapter, aiming to...

R-Ladies Rdam
@RLadiesRdam
R-Ladies chapter in Rotterdam, the Netherland...

R-Ladies Utrecht
@RLadiesUtrecht
R-Ladies Utrecht is part of a world-wide organi...

Brenton Wiernik 🏳️‍🌈 @bmwiernik · 52m

This morning my husband is walking around singing "an object of type Unzi not subsettable". My husband is not an #rstats user. Unzi is a cat.

R Basics

- R: the basic programming language

- Rstudio: an IDE (integrated development environment) for using R

Let’s get cooking!

Scripts & comments

Scripts

- A script is a list of steps for the computer to do
- Scripts (can) contain:
  - Commands
  - Data
  - Comments

Comments

- Comments are indicated by a ‘#’
- Comments make your script readable
  - to others (including us)
  - yourself (10 years from now)

```
# This script analyzes the sleep of students
```

Scripts and Comments

- You should write comments
  - to give structure to long scripts
  - whenever parts of your script are not self-explanatory, e.g., when you define new functions
  - when you are working on assignments

Functions

Functions

- There are a lot of functions available in R
- You can add your own functions
- Functions always have the form:
  print(what = “Hello class”)

Packages

- Contain additional functions and data
- Use install.packages(“packagename”) to install a package
  e.g., install.packages(“ggplot2”)
- Use library(packagename) to load it into your workspace

R objects

Object Modes

- There are different types of objects in R:
    - Numeric
    - Character
    - Logical

- These are called *modes*

- To request the mode of object x use mode(x)

Logical tests

- x == y: Is x *equal* to y?
- x != y: Is x *not equal* to y?
- x > y: Is x *greater than* y?
- x >= y: Is x *greater than or equal* to y?
- x < y: Is x *smaller than* y?
- x <= y: Is x *smaller than or equal* to y?

Testing and Transforming modes

You can use functions named is.*mode* to test the mode of an object:

```
a <-1.23
is.logical(a)
```

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

```
is.numeric(a)
```

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

Testing and Transforming modes

You can use functions named as.*mode* to transform objects into a different mode:

```
"1" + 1
```

```
## Error in "1" + 1: non-numeric argument to binary operat
```

```
as.numeric("1") + 1
```

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

```
as.numeric("abc")
```

```
## Warning: NAs introduced by coercion  ## [1]

NA
```

Missing data

Missing Data

- R encodes missing data as NA (NotAvailable)
- Usually you cannot compute things with NA
- You need to handle NA in a special way

Missing Data

```
x <- NA

# You cannot check for NA like this:
x == NA
```

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

```
# But you can do:
is.na(x)
```

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

NaN and Inf

Special cases of data that often result in problems when computing things are NaN (not a number) and Inf(infinite)

```
# NaN:
0/0
```

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

```
# Inf:
1/0
```

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

NaN and Inf

```r
# Check for NaN:
is.nan(0/0)
```

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

```r
# Check for infinite:
1/0 == Inf
```

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

```r
# Alternatively:
is.infinite(1/0)
```

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

Working directory & workspace

Working Directory

- The location where R ‘looks’ for and stores files
- Use getwd() to see the path to the current working directory
- Use setwd() to change the path to the current working directory

Workspace

- Collection of all objects (data and functions) that are available in the current R session (beside base functions)
- Use ls() to see a list of all objects
- Use rm(list = ls()) to delete all objects

Basic Data Structures

- A data structure is a way of organizing data (information)

Data Structures

object	modes	several modes
vector	numeric, character, or logical	No
matrix	numeric, character, or logical	No
array	numeric, character, or logical	No
list	num, char, logic, function, expressions…	Yes
data frame	numeric, character, or logical	Yes
factor	numeric, or character	No

Vectors

- Vectors are R’s most basic data structure
- A vector is a one dimensional object that stores multiple values of the same mode
- Use c(...) (*concatenate*) to manually create a vector

Vectors

- A regular sequence with a fixed step size can be created using
- seq(*start*,*end*,*step_size*)

- For example, seq(1,7,3) creates the vector 1, 4, 7

Vectors

- A vector with repeated objects be created using

  rep(*object*,*number_of_repetitions*):

- For example, rep(1,3) creates the vector 1,1,1
- Useful variation: rep_len(*object*,*length_out*)

Vectors

- You can combine two or more vectors into one using c(…):
- c(c(1,2), c(3)) gives 1, 2, 3

Vectors

- Draw *n* random samples from a vector using sample(*vector_to_sample_from*, *n*)

Properties of Vectors

- Test the mode of a vector using is.*mode* and tranform its mode using as.*mode*

Properties of Vectors

Test the mode:

```
v <- seq(1,4,0.5)
is.numeric(v)
```

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

Change the mode:

```
v <- seq(1,4,0.5)
as.character(v)
```

```
## [1] "1" "1.5" "2" "2.5" "3" "3.5" "4"
```

Properties of Vectors

- You can test the mode of a vector using is.*mode* and tranform its mode using as.*mode*
- You can check the length of a vector using
  - length(*some_vector*)
- You can initialise vectors of length *n* with a certain mode using
  - numeric(*n*), logical(*n*), character(*n*)

Properties of Vectors

Initialise numeric vector:

```
v <- numeric(3)
v
```

```
## [1] 0 0 0
```

Vector Operations

You can perform elementwise operations on vectors:

```
a <- c(10,20,30)
b <-1:3

a + b # Add each element in a with the same element in
```

```
## [1] 11 22 33
```

b

```
a * b # Multiply each element in a with the same element i
```

```
## [1] 10 40 90
```

Vector Operations

You can perform scalar multiplication on a numeric vector:

```
a <- c(10,20,30)
a * 2 # Multiply all elements in a with 2
```

```
## [1] 20 40 60
```

Vector Operations

You can perform scalar multiplication on a numeric vector:

```
a <- c(10,20,30)
a * 2 # Multiply all elements in a with 2
```

```
## [1] 20 40 60
```

You can perform scalar addition on a numeric vector:

```
a + 2 # Add 2 to all elements in a
```

```
## [1] 12 22 32
```

Vector Operations

You can apply vector multiplication (inner product):

```
a <- c(10,20,30)
b <- c(4,2,4/3)

a%*%b
```

```
##          [,1]
## [1,]      120
```

Vector Operations

You can apply logical operations to vectors:

```
a <- c(10,20,30)
b <- c(5,20,50)

a < 20 # Test each element of a if it is smaller than 20
```

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

Vector Operations

You can apply logical operations to vectors:

```
a <- c(10,20,30)
b <- c(5,20,50)

a < 20 # Test each element of a if it is smaller than 20
```

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

```
a == b # Test if each element in a is equal to the same element in b
```

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

Selecting Vector Elements

- You can select a vector subset using indexing
- Use square brackets [] containing:
  - numbers of the elements you wish to keep
  - a minus sign followed by the element you wish to omit
  - a logical vector indicating whether to keep the specific elements

Selecting Vector Elements

```
a <- c(2,4,6,8,10)
a
```

```
## [1] 2 4  6 8  10
```

```
a[5] # Get the fifth element
```

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

```
a[-5] # Get everything except the fifth element
```

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

Selecting Vector Elements

```
a[c(1,5)] # Get the first and fifth element
```

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

All elements of a that are smaller than 5:

```
a[c(TRUE,TRUE,FALSE,FALSE,FALSE)]
```

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

Selecting Vector Elements

We could get that logical vector with:

```
a < 5
```

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

Thus, this also works!

```
a[a < 5]
```

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

Selecting Vector Elements

```
age <- c(22,20,28,25,32,21,25)
gender <- c('male','male','female','female','male','female’)
```

Selecting Vector Elements

```
age <- c(22,20,28,25,32,21,25)
gender <- c('male','male','female','female','male','female’)

# Age of males:
age[gender == "male"]

## [1] 22 20 32 25
```

```
# Age of females:
age[gender == "female"]

## [1] 28 25 21
```

Potential end of Monday lecture

Potential start of Wednesday lecture

Basic Data Structures
(continued)

- A data structure is a way of organizing data (information)

Data Structures

object	modes	several modes
vector	numeric, character, or logical	No
matrix	numeric, character, or logical	No
array	numeric, character, or logical	No
list	num, char, logic, function, expressions…	Yes
data frame	numeric, character, or logical	Yes
factor	numeric, or character	No

Matrices

- A matrix is vectors “stacked” on top of each other:

```
# a matrix of characters:
matrix(ncol=3,nrow=3, letters[1:9])
```

```
##        [, 1] [, 2] [, 3]
## [1, ] ″a″   ″d″   ″g″
## [2, ] ″b″   ″e″   ″h″
## [3, ] ″c″   ″f″   ″i″
```

Matrices

- A matrix is vectors “stacked” on top of each other:

```
# a matrix of numbers:
matrix(ncol=3,nrow=3, 1:9)
```

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

Matrices

- A matrix is vectors “stacked” on top of each other:

```
# a matrix of characters & numbers?:
matrix(ncol=3,nrow=3, c(1:3,letters[1:6]))
```

```
##       [,1] [,2] [,3]
## [1,] ″1″  ″a″  ″d″
## [2,] ″2″  ″b″  ″e″
## [3,] ″3″  ″c″  ″f″
```

Matrix manipulation

```
a <- matrix(1:9,3,3)
b <- matrix(1:3,1,3)

# Combining
rbind(a,b)
```

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

Matrix manipulation

```
a <- matrix(1:9,3,3)
b <- matrix(1:3,3,1)

#  Combining
cbind(a,b)
```

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

Arrays

- An array is matrices “stacked” behind each other:

```r
array(1:12, dim = c(2,3,2))
```

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

Indexing Matrices/Arrays

```r
a <- matrix(1:9,3,3)
```

- Index the first row:

```r
a[1,]
```

```
## [1] 1 4 7
```

- Index the first column:

```r
a[,1]
```

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

Lists

- The `list()`function can be used to create a *list*
- This is an object that can contain other objects
- To index a list use double square brackets, or a dollar sign

Lists

```
# A vector:
v1 <- c(5,10,1,3)
v1
```

```
## [1]  5 10  1  3
```

```
# A matrix:
m1 <- matrix(c(5, 2, 10, 1),2,2)
m1
```

```
##      [,1] [,2]
## [1,]    5   10
## [2,]    2    1
```

```
# Put them in a list:
l1 <- list(v1 = v1, m1 = m1)
```

Lists

```r
str(l1)
```

```
## List of 2
##  $ v1: num [1:4] 5 10 1 3
##  $ m1: num [1:2, 1:2] 5 2 10 1
```

Lists

```
# Index the vector v1:
l1$v1
```

```
## [1]  5 10  1  3
```

```
l1[['v1']]
```

```
## [1]  5 10  1  3
```

```
# Change an element in the matrix m1:
l1$m1[2,2] <- 0
l1$m1
```

```
##      [,1] [,2]
## [1,]    5   10
## [2,]    2    0
```

Extract results from list

- A lot of functions in R return a `list` as output
- You need to be able to extract certain results from a list
- Let’s say you want the t-statistic from the `t.test()` function

Extract results from list

```r
# Run a t-test:
result <- t.test(1:10, y = 7:20)
```

Extract results from list

```
# Run a t-test:
result <- t.test(1:10, y = 7:20)

# Check the structure of the output:
str(result)
```

```
List of 10
 $ statistic  : Named num -5.43
  ..- attr(*, "names")= chr "t"
 $ parameter  : Named num 22
  ..- attr(*, "names")= chr "df"
 $ p.value    : num 1.86e-05
 $ conf.int   : num [1:2] -11.05 -4.95
  ..- attr(*, "conf.level")= num 0.95
 $ estimate   : Named num [1:2] 5.5 13.5
  ..- attr(*, "names")= chr [1:2] "mean of x" "mean of y"
```

Extract results from list

```
# Select the element you are interested in:
result$statistic
```

```
##         t
## -5.43493
```

```
# Or:
result[["statistic"]]
```

```
##         t
## -5.43493
```

Dataframes

- The `data.frame()`function can be used to create a *dataframe*
- A data frame is a combination of a matrix and a list
  - Looks like a matrix and can be indexed as one
  - Contains *variables* as columns
  - Columns indexed using double brackets or dollar sign
- This is the structure you will use when working with data

Dataframes

```r
# A character vector:
sex <- c("male","female","male","female")

# A logical vector:
exp <- c(TRUE,TRUE,FALSE,FALSE)

# 2 numeric vectors:
A <- c(5,10,1,3)
B <- 1:4

# Put them in a data frame:
df1 <- data.frame(sex=sex,exp=exp,A=A,B=B)
```

Indexing dataframes

```r
# Index the vector sex:
df1$sex
```

```
## [1] male   female male   female
## Levels: female male
```

```r
df1[['sex']]
```

```
## [1] male   female male   female
## Levels: female male
```

```r
# Subset of the data containing only A and B:
df1[,c('A','B')]
```

```
##    A B
## 1  5 1
## 2 10 2
## 3  1 3
## 4  3 4
```

Subsetting dataframes

```
subset(df1, sex == "male")
```

```
##       sex      exp   A B
## 1 male      TRUE   5 1
## 3 male      FALSE 1 3
```

```
subset(df1, A > 3 )
```

```
##          sex     exp    A  B
## 1       male TRUE     5  1
## 2    female TRUE    10 2
```

Structure of dataframes

```r
str(df1)
```

```
## 'data.frame':        4 obs. of      4 variables:

##    $ sex: Factor w/ 2 levels "female","male": 2 1 2 1
##    $ exp: logi        TRUE TRUE FALSE FALSE
##    $ A    : num      5 10 1 3
##    $ B    : int       1 2 3 4
```

Factors

- Factors encode categorical variables
- Consists of levels, some of which might not be in your data
- Labels are characters but levels correspond to numbers!

Factors

```
# For 5 people, I measured the city they live in:
city <- factor(c("Amsterdam","London","Amsterdam",
                        "Brussels","London"))

# Looks like a character vector but gives levels:
city
```

```
## [1] Amsterdam London    Amsterdam Brussels  London
## Levels: Amsterdam Brussels London
```

```
# Indexing one still tells me all possible outcomes:
city[1]
```

```
## [1] Amsterdam
## Levels: Amsterdam Brussels London
```

Factors

```
# There can be more outcomes than I have
city <- factor(c("Amsterdam","London","Amsterdam",
                 "Brussels","London"),
        levels = c("Amsterdam", "London", "Paris", "Brussels"))

city

## [1] Amsterdam London    Amsterdam Brussels  London
## Levels: Amsterdam London Paris Brussels
```

```
# Converting to numeric gives me integers:
as.numeric(city)

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

Factors

```
# WARNING! In data frames (discussed next)
# character vector are changed into factors!
df <- data.frame(city = c("amsterdam", "london"))
df$city
```

```
## [1] amsterdam london
## Levels: amsterdam london
```

Factors

```
# WARNING! If a factor looks numeric it is still a factor!
# as.numeric changes to levels:
foo <- factor(c('4','1','10'))
foo
```

```
## [1] 4  1  10
## Levels: 1 10 4
```

```
as.numeric(foo)
```

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

```
as.numeric(as.character(foo))
```

```
## [1] 4  1  10
```

End of data structures

[Reading and writing files in R]

Ways of saving/loading data

read.csv() and write.csv() can save data frames or matrices.

Ways of saving/loading data

read.csv() and write.csv() can save data frames or matrices.

You can save any R objects (e.g., lists, functions) using save(*object*,*file_name.RData*).

Using this function saves objects in .Rdata files

.Rdata files can be loaded into R using the load() function

```
save(bigObject,file ="bigObject.Rdata")
load(file ="bigObject.Rdata")
```

String Manipulation

String Manipulation

It is often handy to automatically generate strings with a certain structure

Use paste(*strings and objects*, sep = *separator*) to concatenate strings:

```r
a <- paste("sub",1:3,sep ="_")
a
```

```
## [1] "sub_1" "sub_2" "sub_3"
```

String Manipulation

You can use grep(pattern = *regexp*, x = *some string*) to search for patterns in a string object:

```
b <- paste("some","other","string",sep ="_")
b
```

```
## [1] "some_other_string"
```

```
grep(pattern ="_", b)
```

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

```
grep(pattern ="blah", b)
```

```
## integer(0)
```

String Manipulation

You can use gsub(pattern = *regexp*, replacement = *regexp*, x = *some string*) to replace a pattern with another pattern

```r
gsub(pattern ="some_",replacement ="", b) # delete 'some_'
```

```
## [1] "other_string"
```

- Another useful one:
  strsplit(x = *some string*, split = *regexp*)

Regular Expressions

- Are a way of searching for patterns in strings
- Are strings extended with a set of symbols

- | Boolean ‘or’
- ! Boolean negation
- . match a single character of any value (except eol)
- * match 0 or more times
- + match 1 or more times
- ˆ match start of the line
- $ match end of the line
- [] match any single character from within the bracket

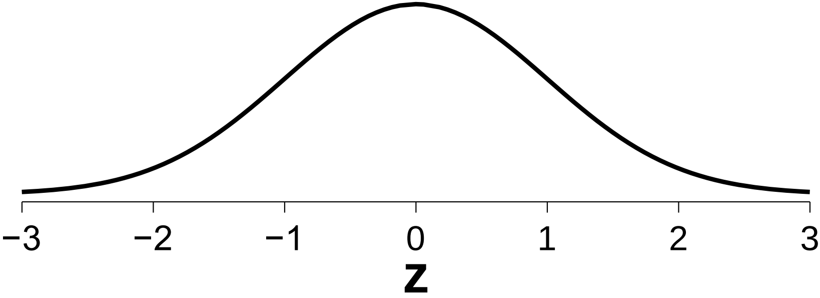
https://www.jdatalab.com/data_science_and_data_mining/2017/03/20/regular-expression-R.html

Generating (Pseudo-)Random Data

Generating Data

Generating data from a normal distribution:

```
rnorm(50,mean = 100,sd = 10)
[1]  93.75615   114.89583   108.98011   107.18434
[5]  92.87754    99.97978    96.76656    108.33115 ...
```

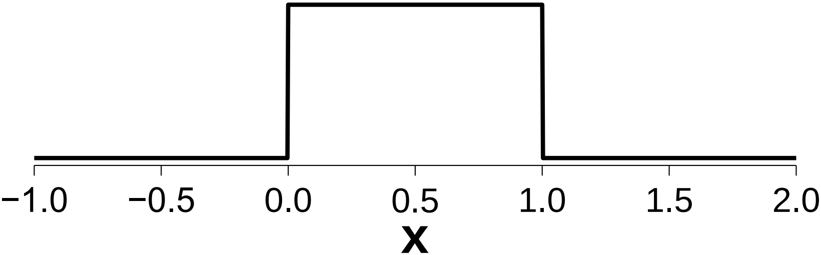


Generating Data

Generating data from a uniform distribution:

```r
runif(50,min = 0,max = 1)
```

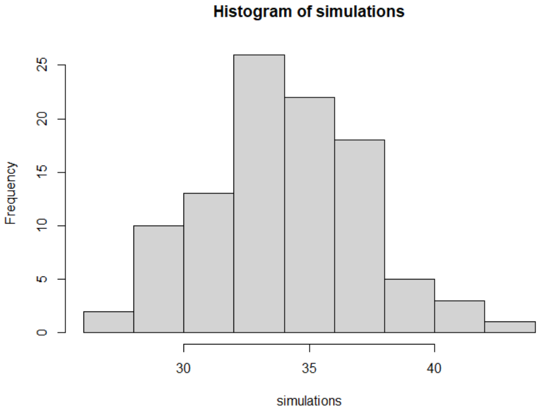
```
##     [1] 0.81326149 0.47378835 0.54834267 0.07856988
##     [5] 0.97161440 0.45413175 0.13792212 0.23808200...
```



Generating Data

Generating data from a normal distribution:

```
simulations <- rbinom(100, size = 50, prob =0.7)
hist(simulations)
```



Other distributions

- Poisson: rpois(*n, lambda*)
- Exponential: rexp(*n, rate*)
- Lognormal: rlnorm(*n, meanlog,sdlog*)
- Gamma: rgamma(*n, shape, scale*)
- Beta: rbeta(*n, shape1,shape2*)
- T: rt(*n, df*)
- F: rf(*n, df1, df2*)
- Chi-Square: rchisq(*n,df*)

Getting Help

- Use the R help functions ? and ??
- Use **Google**
- Use stackexchange:
  - For programming technical questions: http://stackoverflow.com/
  - For statistical questions: http://crossvalidated.com/
  - Use the tag ‘r’ and include a reproducable example (http://stackoverflow.com/q/5963269/567015)
- Blogs on R are available at: http://www.r-bloggers.com/

Do you remember everything I said?

- (No)
- You only learn coding by coding

  → Assignment