Programming with R

Alireza Soltani Khaboushan
MD-MPH
Pediatric Urology and Regenerative Medicine Research Center
Tehran University of Medical Sciences

```
Me = {      "Name" : "Alireza Soltani",
            "Age" : 22,
            "Education" : [
                        "MD student at TUMS",
                        "Cardiology Research Diploma at THC",
                        "MPH student at TUMS"
            ],
            ”Roles" : [
                        "Research Assistant at PURMRC",
                        "Structural Editor at JOST”,
                        "Referee at SSRC Research Council”
            ],
            "Interests" : [
                        "Neurology",
                        "Regenerative Medicine",
                        "Programming",
                        "Statistics",
                        "Artificial Intelligence"
            ],
            "Skills" : [
                        # There’s always room for improvement:)
            ],

            "E-mail" : "alirezasoltanykhaboshan@gmail.com",
            ”LinkedIn" : "https://www.linkedin.com/in/alireza-soltani-709848255/",
            ”ResearchGate" : "https://www.researchgate.net/profile/Alireza-Soltani-Khaboushan",
}
```

Then, **R** you familiar with **R**?

TABLE OF CONTENTS

01 Why R?

02 Basics of Coding with R

03 Descriptive Statistics and Visualization

04 Inferential Statistics

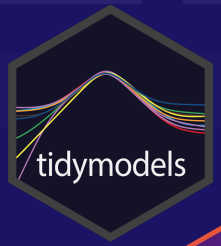
01

Why R?

R was developed by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, in the mid-1990s, and is now maintained by the R Development Core Team.

The initial version of R was released in 1995, and it was modeled after the S language, which was developed at Bell Laboratories by John Chambers and his colleagues.
R has a wide range of functionalities for statistical analysis, data visualization, and data manipulation.

It is widely used in both academia and industry, particularly in fields such as data science, finance, and biostatistics.

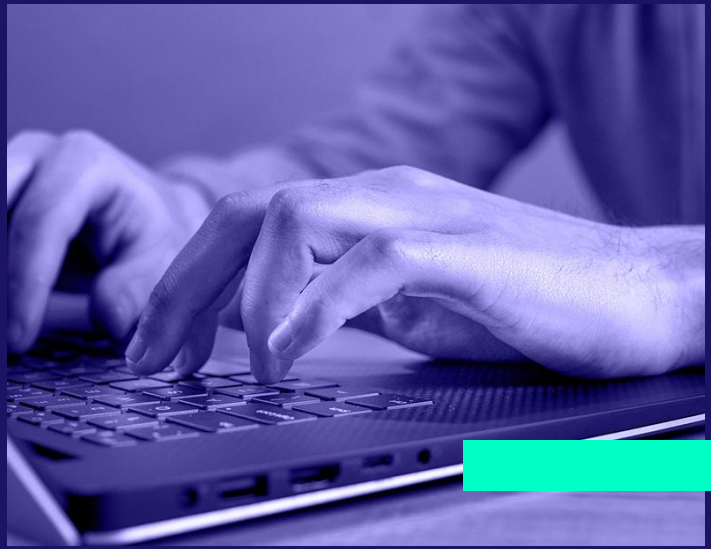


WHY R?

Is it still worth learning R?
Can R be replaced by python?
Is it a dying language?
R vs Stata vs MATLAB?

Pros

Free
Open-Source
Visualization
Multiple Cutting-Edge Statistical Packages
Data Handling and Manipulation
Meta-Analysis
Integrated Development Environment
Help Reproducible Research
Shiny, dplyr, and ggplot2



Cons

Not Optimized Usage of Memory
Indexing Starts from One Instead of Zero
Inconsistent Packages and Syntaxes
Could Get Dreadfully Slow
Better Alternatives Available for Deep Learning
Not the Best Option for Object-Oriented Programming

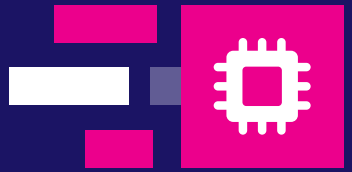
Download and Install R and Rstudio

CRAN

https://cran.rstudio.com/

Rstudio

https://posit.co/download/rstudio-desktop/



Download and install

Choose your OS and install R

Download and install

Choose your OS and install RStudio

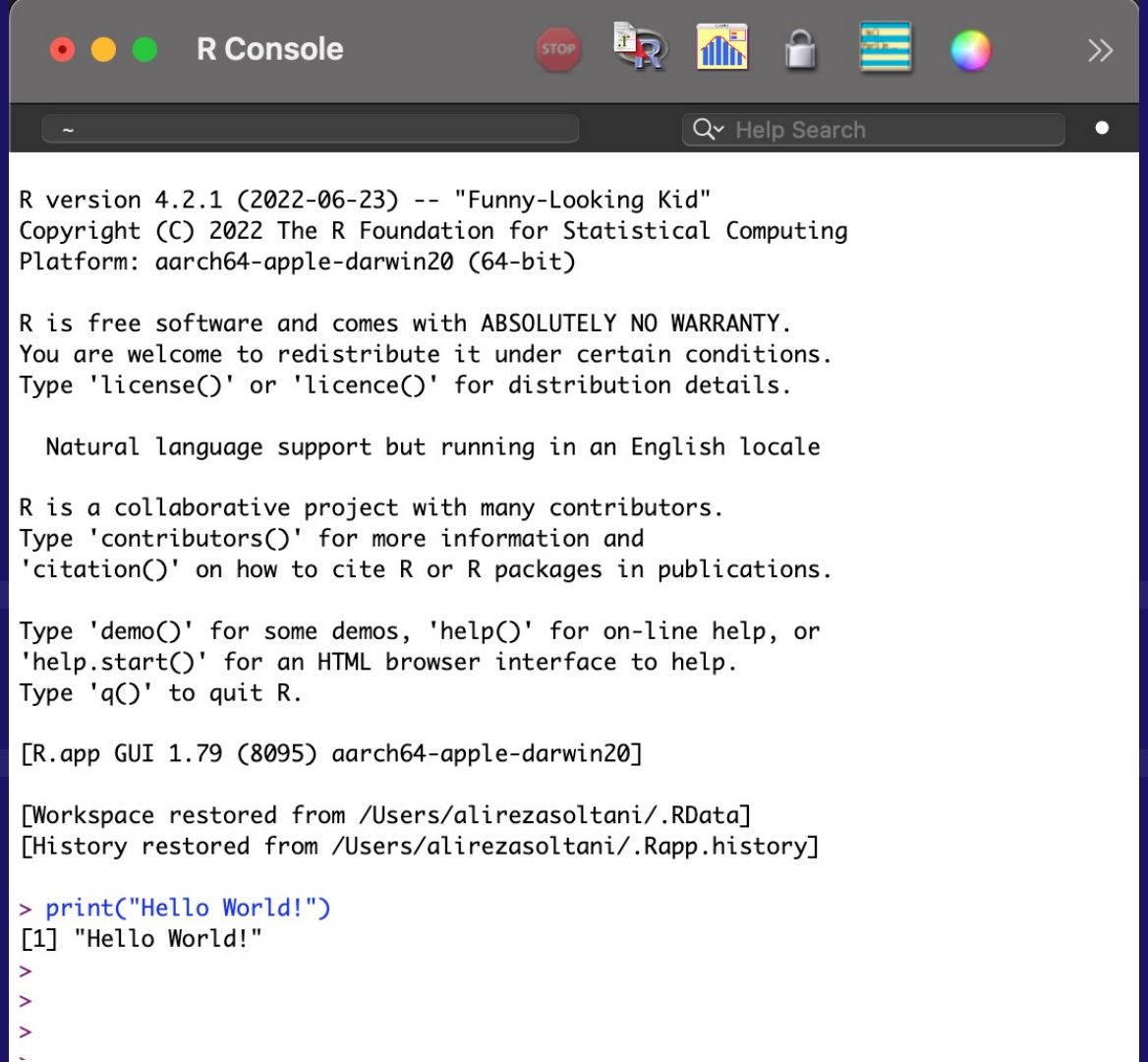
Learning R

ChatGPT

GitHub Copilot

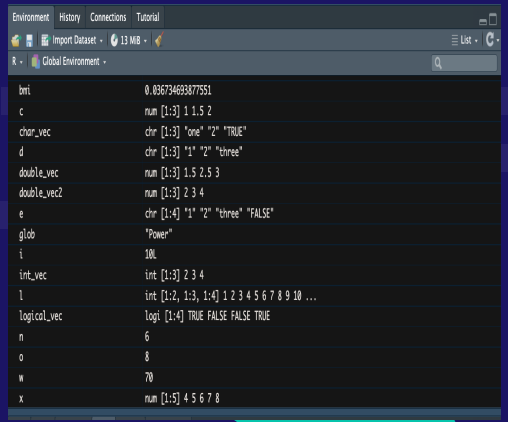
02

Basics of Coding with R

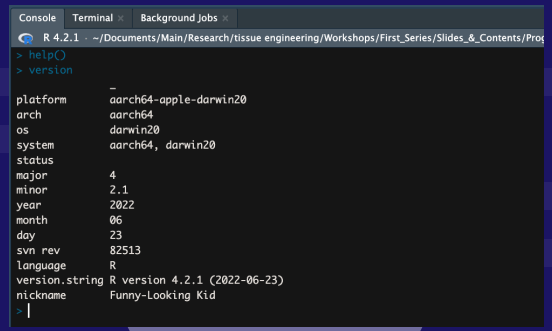

R Console
STOP
~
Help Search
R version 4.2.1 (2022-06-23) -- "Funny-Looking Kid"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
[R.app GUI 1.79 (8095) aarch64-apple-darwin20]
[Workspace restored from /Users/alirezasoltani/.RData]
[History restored from /Users/alirezasoltani/.Rapp.history]
> print("Hello World!")
[1] "Hello World!"
>
>
>
>

RStudio® Environment

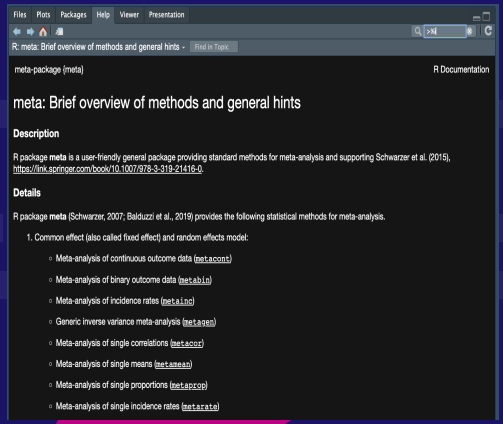
Environment

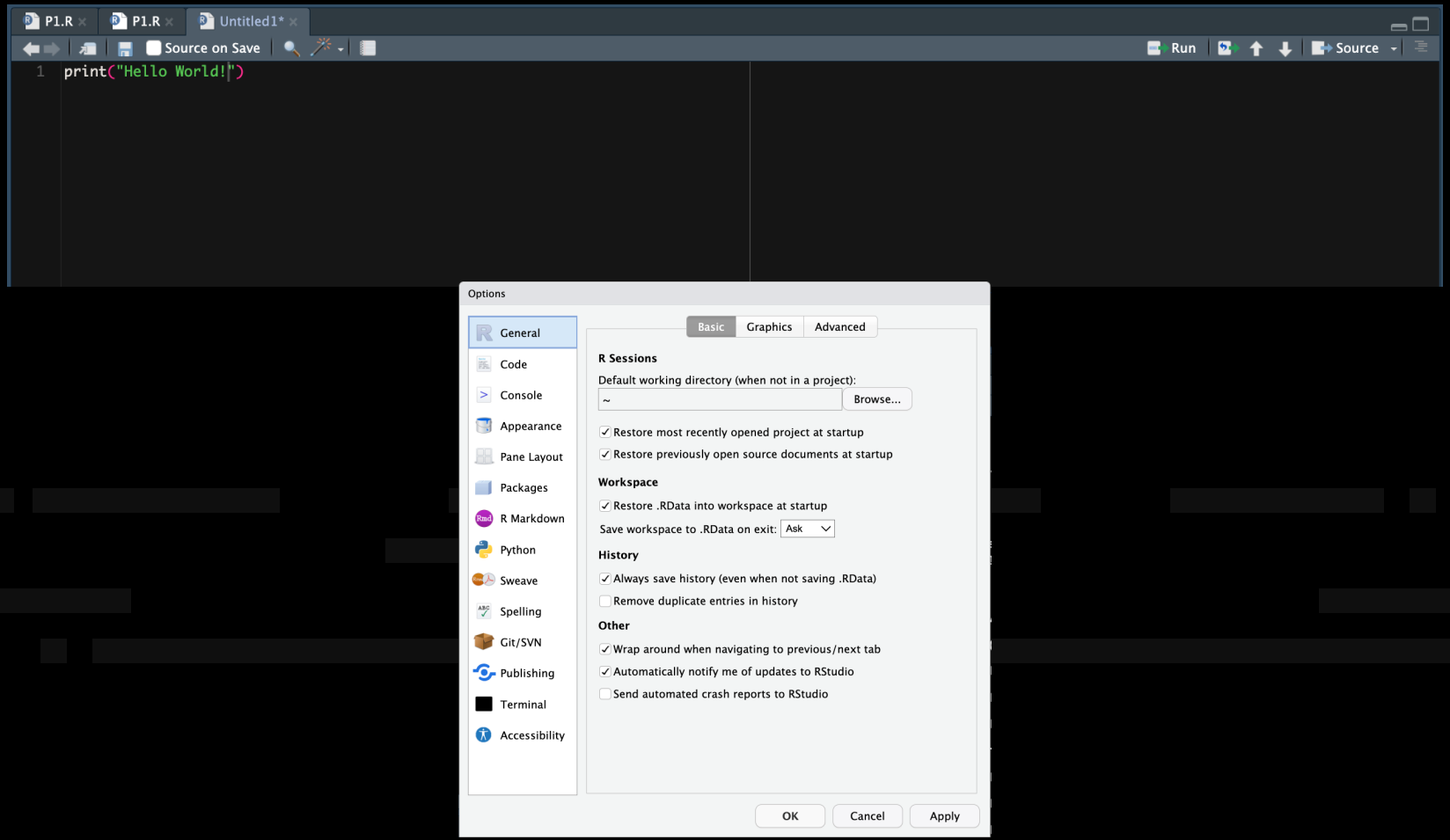


Console



Outputs, packages, etc.



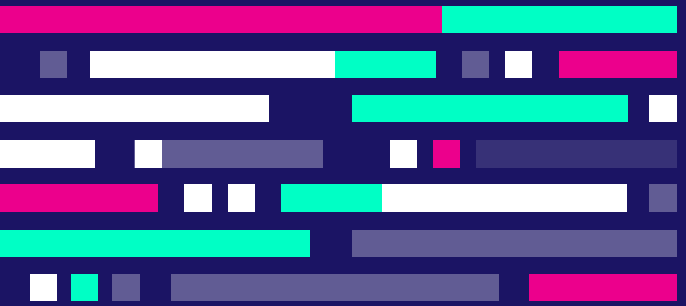

P1.R
P1.R
Untitled1*
Source on Save
Run
Source
1
print("Hello World!")
Options
General
Code
Console
Appearance
Pane Layout
Packages
R Markdown
Python
Sweave
Spelling
Git/SVN
Publishing
Terminal
Accessibility
Basic
Graphics
Advanced
R Sessions
Default working directory (when not in a project):
~
Browse...
Restore most recently opened project at startup
Restore previously open source documents at startup
Workspace
Restore .RData into workspace at startup
Save workspace to .RData on exit:
Ask
History
Always save history (even when not saving .RData)
Remove duplicate entries in history
Other
Wrap around when navigating to previous/next tab
Automatically notify me of updates to RStudio
Send automated crash reports to RStudio
OK
Cancel
Apply

Getting Started with R

```
> print("Hello World!")
[1] Hello World!
```

- Arithmetic Operations

+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Power
%%	Remainder
%/%	Quotient
%*%	Matrix Multiplication
sqrt()	Square Root
abs()	Absolute Value



Operator Syntax and Precedence

:: :::	access variables in a namespace
$ @	component / slot extraction
[[[	indexing
!	negation
& &&	and (bit-wise and Boolean, respectively)
| ||	or (bit-wise and Boolean, respectively)
~	as in formulae
-> ->>	rightwards assignment
<- <<-	assignment (right to left)
=	assignment (right to left)
?	help (unary and binary)
%>%	pipe
%in%	value matching

```
> 5 + 2
[1] 7
> 5 - 2
[1] 3
> 5 * 2
[1] 10
> 5 / 2
[1] 2.5
> 5 ^ 2
[1] 25
> 5 %% 2
[1] 1
> 5 %/% 2
[1] 2
> sqrt(9)
[1] 3
> abs(-9)
[1] 9
```

Assignment

- To assign value to variable use <- instead of =

```
> h <- 180
> w <- 70
> h
[1] 180
> w
[1] 70
```

- To assign arguments in the function use =

```
> rnorm(n = 2, mean = 5, sd = 2)
[1] 2.567296 5.100900
```

Example

Calculate BMI

```
12  h <- 180
13  w <- 70
14  bmi <- h / (w ^ 2)
> bmi
[1] 0.03673469
```

Variables and Data Types

- Variable stores specific data which could be in various forms:

1. Integer — 5:100, 5L
2. Double — 5.0, 5, c(1, 2, 3, 4)
3. Logical — True, T, False, F
4. Character — “True”, “Name”
5. List — list(1, 2, 3)
6. Function — a <- fun(x) {mean(1:x)}
7. Raw — raw(2)
8. Complex — 2+0i

- Check the type of variables using typeof() function

Variables and Data Types

- A variable name must start with a letter and can be a combination of letters, digits, period(.), and underscore(_).
- If it starts with period(.), it cannot be followed by a digit.
- A variable name cannot start with a number or underscore (_).
- Variable names are case-sensitive (age, Age and AGE are three different variables).
- Reserved words cannot be used as variables (TRUE, FALSE, NULL, if...).
- Variables name are snake_case in R.

Example

```
32  ali <- 12
33  .2 <- "hi"
34  _3 <- "yes"
35  _so <- 2 * 6
36  2a <- "salaam"
37  As2 <- "congrats"
38  absolute_value <- abs(-18)  # snake_case
39  absolute-value <- abs(-20)  # kebab-case
40  absoluteValue <- abs(-25)   # camelCase
41  AbsoluteValue <- abs(-83)   # PascalCase
> ali <- 12
> .2 <- "hi"
Error in 0.2 <- "hi" : invalid (do_set) left-hand side to assignment
> _3 <- "yes"
Error: unexpected numeric constant in "_3"
> _so <- 2 * 6
Error: unexpected symbol in "_so"
> 2a <- "salaam"
Error: unexpected symbol in "2a"
> As2 <- "congrats"
> absolute_value <- abs(-18)  # sanke_case
> absolute-value <- abs(-20)  # kebab-case
Error in absolute - value <- abs(-20) : object 'absolute' not found
> absoluteValue <- abs(-25)   # camelCase
> AbsoluteValue <- abs(-83)   # PascalCase
```

Conditionals

- If, else if, and else are used for conditional statements

```
32 ▾ if(h > 165) {
33     print("Normal")            # if height is higher than 165 "Normal is printed
34 ▾ } else if (h == 165) {
35     print("threshold")         # if height is equal to 165 "threshold" is printed
36 ▾ } else {
37     print ("short")            # if height is lower than 165 "short" is printed
38 ▴ }
> h <- 165
> if(h > 165) {
+   print("Normal")            # if height is higher than 165 "Normal is printed
+ } else if (h == 165) {
+   print("threshold")         # if height is equal to 165 "threshold" is printed
+ } else {
+   print ("short")            # if height is lower than 165 "short" is printed
+ }
[1] "threshold"
```

- # is used to add comments to the code

Comments will help the readability and reproducibility of the code

ifelse()

- A handy-dandy function help with applying conditions on vectors, lists, data frames
- It prevents using excessive for loop and speeds up the code

```
ifelse(test, yes, no)
x <- c(6:-4)
sqrt(x)  #- gives warning
sqrt(ifelse(x >= 0, x, NA))  # no warning
```

Functions

- Functions hugely help to do a work repeatedly

```
function(arguements){
  body()
}
# Function to calculate the BMI by getting the weight and height
BMI <- function(height, weight, unit = "cm") {
  if (unit != "cm"){
    print("the height should be in cm")
  } else {
  bmi <- weight / (height ^ 2)
  print(bmi)
  }
}
```

- Calling Functions
- Default Arguments
- Global variables
- Nested Functions
- Recursive functions

Example

- Calling Functions and Default Arguments

```
> BMI(180, 120)
[1] 0.003703704
> BMI(height = 180, weight = 120)
[1] 0.003703704
> BMI(height = 180, weight = 120, unit = "m")
[1] "the height should be in cm"
> bmi(180, 120)
Error in bmi(180, 120) : could not find function "bmi"
```

Example

- Global Variables

```
 97  # global variable
 98  glob <- "Hello"
 99  glob_fun <- function(){
100    paste(glob, "World!")
101  }
102  glob_fun()
103
104  glob_fun2 <- function(){
105    glob <- "Damn"
106    paste(glob, "World!")
107  }
108  glob
109  glob_fun2()
110
111  glob_fun3 <- function(){
112    glob <<- "Power"
113  }
114  glob
115  glob_fun3()
116  glob
> glob
[1] "Hello"
> glob_fun2()
[1] "Damn World!"
> glob_fun3 <- function(){
+   glob <<- "Power"
+ }
> glob
[1] "Hello"
> glob_fun3()
> glob
[1] "Power"
```

Example

- Nested Function

```r
# Nested function
outer_func <- function(x) {
  inner_func <- function(y) {
    a <- x + y
    return(a)
  }
  return (inner_func)
}
output <- outer_func(2) # To call the Outer_function
output(6)
```

```
> output <- outer_func(2) # To call the Outer_function
> output(6)
[1] 8
```

- Recursive Function

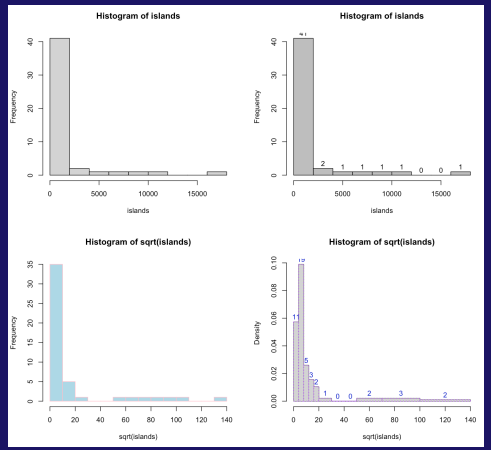
```r
# Recursive functions
recursion <- function(x) {
  if (x > 1) {
    result <- x + recursion(x - 1)
    print(result)
  } else {
    result = 0
    return(result)
  }
}
recursion(6)
```

```
> recursion(6)
[1] 2
[1] 5
[1] 9
[1] 14
[1] 20
```

- help(“package”)

To have a description about the package and its application

- example(“hist”)



Loops (For Loops)

- "For Loops" are used for repetitious doing of a task

```
For (iterator in list or vector) {
        statement
}
```

- For loop could be combined with conditionals
- With the next statement, we can skip an iteration without terminating the loop
- With the break statement, we can stop the loop before it has looped through all the items
- A for loop could be inside another for loop as nested for loop

Example

- For loop

```r
# for loop
for (i in 1:10){
  a = rep(x = "*", i)
  print(a)
}
```

```
[1] "*"
[1] "*" "*"
[1] "*" "*" "*"
[1] "*" "*" "*" "*"
[1] "*" "*" "*" "*" "*"
[1] "*" "*" "*" "*" "*" "*"
[1] "*" "*" "*" "*" "*" "*" "*"
[1] "*" "*" "*" "*" "*" "*" "*" "*"
[1] "*" "*" "*" "*" "*" "*" "*" "*" "*"
 [1] "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
```

- For loop and conditionals

```r
m <- list("apple", "banana", "pitch", "strawberry", "melon", "orange")
for (i in m){
  if (i == "pitch"){
    next
  } else if (i == "melon") {
    break
  } else {
    print(i)
  }
}
```

```
[1] "apple"
[1] "banana"
```

While

- With the while loop we can execute a set of statements as long as a condition is TRUE

```r
# while
n <- 1
while (n < 6) {
  print(n)
  n <- n + 1
}
```

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

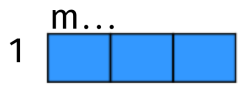
- Repeat loop in R is used to iterate over a block of code multiple number of times. And also it executes the same code again and again until a break statement is found.

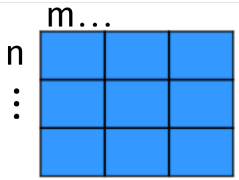
```r
o <- 2
repeat {
  print(o)
  o = o + 3
  if (o == 8) {
    break
  }
}
```

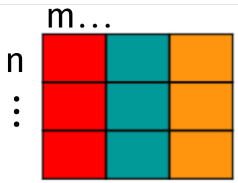
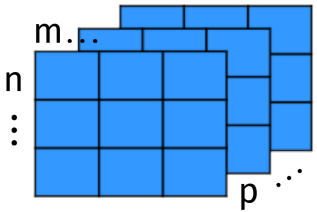
```
[1] 2
[1] 5
```

Data Structures

	Homogenous	Heterogenous
1-Dimensional	Vector	List
2-Dimensional	Matrix	Data Frame
n-Dimensional	Array	

	Dimensions	Mode (data "type")	Example
Vector		Identical	`c(10,0.2,34,48,53)`

Matrix		Identical	`matrix(c(1,2,3,`
			`11,12,13),`
			`nrow = 2,`
			`ncol = 3)`

Data frame		Can be different	`data.frame(x = 1:3,` `y = 5:7)`
Array		Identical	`array(data = 1:3,` `dim = c(2,4,2))`
List	Vector Matrix Data frame Array	Can be different	`list(x = cars[,1],` `y = cars[,2])`

Some practical functions for interrogating data structure in R

str()
class()
typeof()
summary()
attributes()
names()
dim()
length()
nrow()
ncol()
colnames()
…

Str() vs class() vs typeof()

In summary, str provides a detailed summary of an object's structure, class returns the class of an object which defines its type, and typeof returns the basic data type of an object.

Vector

```
126  a <- 1
127  b <- "b"
> str(a)
 num 1
> str(b)
 chr "b"
> class(a)
[1] "numeric"
> class(b)
[1] "character"
> typeof(a)
[1] "double"
> typeof(b)
[1] "character"
> is.vector(a)
[1] TRUE
> is.atomic(a)
[1] TRUE
> is.vector(b)
[1] TRUE
> is.atomic(b)
[1] TRUE
```

Vector

Vectors (atomic vectors) are usually created with C():

```
135  double_vec <- c(1.5, 2.5, 3.0)
136  double_vec2 <- c(2, 3, 4)
137  int_vec <- c(2L, 3L, 4L) # L suffix creates integer rather than double
138  char_vec <- c("one", "2", "TRUE")
139  logical_vec <- c(TRUE, FALSE, F, T)
> typeof(double_vec)
[1] "double"
> typeof(double_vec2)
[1] "double"
> typeof(int_vec)
[1] "integer"
> typeof(char_vec)
[1] "character"
> typeof(logical_vec)
[1] "logical"
```

Vector

```
135  double_vec <- c(1.5, 2.5, 3.0)
136  double_vec2 <- c(2, 3, 4)
137  int_vec <- c(2L, 3L, 4L) # L suffix creates integer rather than double
138  char_vec <- c("one", "2", "TRUE")
139  logical_vec <- c(TRUE, FALSE, F, T)
> is.numeric(double_vec2)
[1] TRUE
> is.vector(double_vec2)
[1] TRUE
> is.atomic(double_vec2)
[1] TRUE
> is.double(double_vec2)
[1] TRUE
> is.integer(double_vec2)
[1] FALSE
> is.character(double_vec2)
[1] FALSE
> is.character(char_vec)
[1] TRUE
> is.logical(logical_vec)
[1] TRUE
```

Vector

Coercion

- Atomic vector only can accept values of one type; thus, if there are various values it coerce them to become one type.

```
161  # Coercion
162  c <- c(1L, 1.5, 2L)
163  d <- c(1, "2", "three")
164  e <- c(1, "2", "three", FALSE)
> typeof(c)
[1] "double"
> typeof(d)
[1] "character"
> typeof(e)
[1] "character"
```

List

Lists could be constructed using list() function:

```
169  # list
170  f <- list(
171    1,
172    "blah",
173    14:20,
174    seq(1, 7, 2),
175    c("one", "two", "three"),
176    c(FALSE, T)
177  )
> f
[[1]]
[1] 1

[[2]]
[1] "blah"

[[3]]
[1] 14 15 16 17 18 19 20

[[4]]
[1] 1 3 5 7

[[5]]
[1] "one"   "two"   "three"

[[6]]
[1] FALSE  TRUE
```

```
> str(f)
List of 6
 $ : num 1
 $ : chr "blah"
 $ : int [1:7] 14 15 16 17 18 19 20
 $ : num [1:4] 1 3 5 7
 $ : chr [1:3] "one" "two" "three"
 $ : logi [1:2] FALSE TRUE
> class(f)
[1] "list"
> typeof(f)
[1] "list"
```

Recursive lists and Coercion

```
191  # Coercion
192  i <- list(
193    c(1, 3),
194    list("a", T)
195  )
196
197  j <- c(
198    c(1, 3),
199    list("a", T)
200  )
> str(i)
List of 2
 $ : num [1:2] 1 3
 $ :List of 2
  ..$ : chr "a"
  ..$ : logi TRUE
> str(j)
List of 4
 $ : num 1
 $ : num 3
 $ : chr "a"
 $ : logi TRUE
```

Matrix and Array

- Arrays are formed when dimensions are added to a vector
- Matrices are a specific type of arrays with only two dimensions
- Arrays and matrices could be defined using, array() and matrix() functions, respectively.

Matrix and Array

```r
# arrays and matrices
k <- matrix(data = 1:24, nrow = 4, ncol = 6)
l <- array(data = 1:24, dim = c(2, 3, 4))
k_prime <- array(data = 1:24, dim = c(4, 6))
str(k)
str(l)
is.matrix(k)
is.matrix(l)
is.array(k)
is.array(l)
is.matrix(k_prime)
all(k_prime == k)
```

```
> str(k)
 int [1:4, 1:6] 1 2 3 4 5 6 7 8 9 10 ...
> str(l)
 int [1:2, 1:3, 1:4] 1 2 3 4 5 6 7 8 9 10 ...
> is.matrix(k)
[1] TRUE
> is.matrix(l)
[1] FALSE
> is.array(k)
[1] TRUE
> is.array(l)
[1] TRUE
> k_prime <- array(data = 1:24, dim = c(4, 6))
> is.matrix(k_prime)
[1] TRUE
> is.matrix(k_prime)
[1] TRUE
> all(k_prime == k)
[1] TRUE
```

```
> k
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
> l
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

, , 3

     [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18

, , 4

     [,1] [,2] [,3]
[1,]   19   21   23
[2,]   20   22   24
```

Matrix and Array

- %*% could be used for matrix multiplication
- t() function is used to transpose the matrix
- dim() gives the dimensions of the matrix

```
> dim(k)
[1] 4 6
> t(k)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16
[5,]   17   18   19   20
[6,]   21   22   23   24
> k %*% t(k)
     [,1] [,2] [,3] [,4]
[1,] 1006 1072 1138 1204
[2,] 1072 1144 1216 1288
[3,] 1138 1216 1294 1372
[4,] 1204 1288 1372 1456
```

Data Frame

Data Frame

- The most common and important method for storing data in R
- It has a 2D structure
- Generally it is a list of vectors with identical structure
- Data frame is created using data.frame() function
- Data frame items could be accessed through the following indices:

```
df[row number, column number]
df[row number, column name]
df$column name[row number]
```

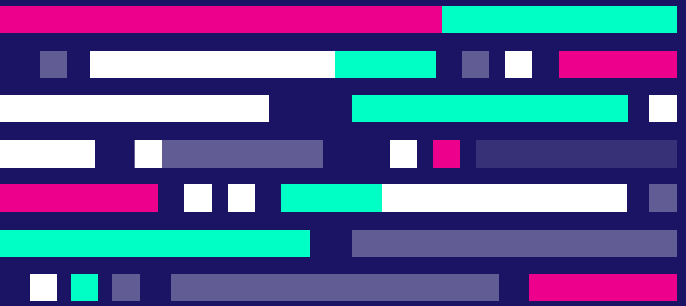
Example

Data Frame

```r
df <- data.frame(
  No = 1:10,
  Odds = log(seq(1, 20, 2)),
  weights = runif(n = 10, min = 50, max = 100),
  hieght = runif(n = 10, min = 150, max = 210),
  marital_status = sample(c("S", "M"), size = 10, replace = T)
    )
df
summary(df)
str(df)
class(df)
typeof(df)
```

```
> summary(df)
       No             Odds          weights          hieght      marital_status
 Min.   : 1.00   Min.   :0.000   Min.   :57.77   Min.   :151.4   Length:10
 1st Qu.: 3.25   1st Qu.:1.694   1st Qu.:66.27   1st Qu.:176.1   Class :character
 Median : 5.50   Median :2.298   Median :83.09   Median :181.9   Mode  :character
 Mean   : 5.50   Mean   :2.030   Mean   :77.71   Mean   :178.2
 3rd Qu.: 7.75   3rd Qu.:2.672   3rd Qu.:88.10   3rd Qu.:185.4
 Max.   :10.00   Max.   :2.944   Max.   :89.83   Max.   :193.1
> str(df)
'data.frame':   10 obs. of  5 variables:
 $ No            : int  1 2 3 4 5 6 7 8 9 10
 $ Odds          : num  0 1.1 1.61 1.95 2.2 ...
 $ weights       : num  87.3 80.4 76.5 62.8 89.8 ...
 $ hieght        : num  151 180 188 186 161 ...
 $ marital_status: chr  "S" "M" "M" "M" ...
> class(df)
[1] "data.frame"
> typeof(df)
[1] "list"
```

Installing Packages



- From CRAN
  - install.packages("package name")
- From github
  - install.packages("devtools")
  - devtools::install_github("package name")
- Library(package name) or require(package name) are used to load package
- Library() to check all installed packages and sessionInfo() to check all loaded packages
- citation() to get reference for specific package

dplyr and tidyverse packages

- install.packages("dplyr")
- library(dplyr)

Or to have all needed basic packages

- install.packages("tidyverse")
- library(tidyverse)

```
> library(tidyverse)
── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──
✔ ggplot2 3.4.0      ✔ purrr   1.0.1
✔ tibble  3.1.8      ✔ dplyr   1.0.10
✔ tidyr   1.2.1      ✔ stringr 1.4.1
✔ readr   2.1.3      ✔ forcats 0.5.2
── Conflicts ────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
```

Read and write CSV

- setwd() is used to set the default directory to what we are going to work with
- getwd() is used to get the current working directory
- readr::read_csv("file path")
- readr::write_csv(x = data, file = "file path")

Filter, Select, Subset

- filter() is used to apply conditionals on rows of a column
- select() is used to select or remove a specific column
- subset() function can do both jobs at the same time

Example

Filter, Select, and Subset

```
298  # filter and select
299  df_2 <- filter(df_1, height > 170)
300  df_3 <- df_1 %>%
301    filter(height > 170)
302  df_3 <- df_1 %>%
303    filter(df_1$height > 170)
304
305  df_4 <- select(df_1, -male_female)
306  df_5 <- select(df_1, -c(male_female, id))
307  df_6 <- df_1 %>%
308    select(-c(male_female, id))
309
310  df_7 <- df_1 %>%
311    filter(height > 167) %>%
312    select(-c(male_female))
313
314  df_8 <- subset(df_1, height > 167, -c(male_female))
```

Example

Subseting without using functions

```
316 df_9 <- df_1[which(df_1$height > 167), -c(5)]
317 df_10 <- df_1[which(df_1$height > 167), -c(which(colnames(df_1) == "male_female"))]
318
319 df_11 <- arrange(df_1, height)
320 df_12 <- arrange(df_1, desc(height))
```

Mutate and Transmute

- mutate() is used to change an existing column in a data frame
- transmute() is used to define a new column in a data frame

```
322  #transmute and mutate
323  df_13 <- df_1 %>%
324    select(-male_female) %>%
325    mutate(male_female = factor(gender, levels = c(0, 1), labels = c("male", "female")))
326
327  df_14 <- df_1 %>%
328    transmute(male_female = factor(gender, levels = c(0, 1), labels = c("male", "female")))
```

Factor

Factors are used to categorize data. Examples of factors are:

- Demography: Male/Female
- Music: Rock, Pop, Classic, Jazz
- Training: Strength, Stamina

```
> fac
 [1] female male   male   male   female male   male   female female female male
[12] male   female male   female female female
Levels: male female
> str(fac)
 Factor w/ 2 levels "male","female": 2 1 1 1 2 1 1 2 2 2 ...
> class(fac)
[1] "factor"
> typeof(fac)
[1] "integer"
```

03

Basics of Statistics with R

Mean, Median, Mode

- The mean is useful for understanding the central tendency of a data set.
- The median is useful for understanding the typical value in a data set and is less sensitive to extreme values than the mean.
- The mode is useful for understanding the most common value in a data set and can be helpful in identifying patterns or trends.

Example

Mean, Mode, and Median

```
> head(df_s4)
  education age parity induced case spontaneous stratum pooled.stratum
1    0-5yrs  26      6       1    1           2       1              3
2    0-5yrs  42      1       1    1           0       2              1
3    0-5yrs  39      6       2    1           0       3              4
4    0-5yrs  34      4       2    1           0       4              2
5   6-11yrs  35      3       1    1           1       5             32
6   6-11yrs  36      4       2    1           1       6             36
390  mean(df_s4$age)
391  median(df_s4$age)
392  mode <- unique(df_s4$parity)
393  mode[which.max(tabulate(match(df_s4$parity, q)))]
394  table(df_s4$parity)
> mean(df_s4$age)
[1] 31.50403
> median(df_s4$age)
[1] 31
> mode <- unique(df_s4$parity)
> mode[which.max(tabulate(match(df_s4$parity, q)))]
[1] 1
> table(df_s4$parity)

 1  2  3  4  5  6
99 81 36 18  6  8
```

Some other useful functions to check and describe data

- min() and max()
- sd()
- IQR()
- range()
- length()
- unique()
- quantile()
- summary()
- Hmisc::describe()
- psych::describe()
- table()
- table.prop()
- distinct()
- gather()

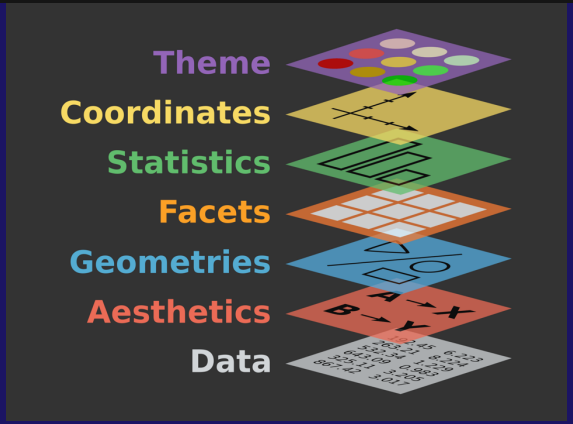
Group_by

To provide summary statistics based on different groups of a variable existing in a database

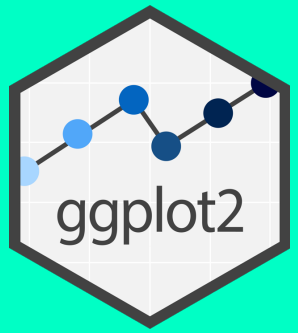
```
403  df_s4 %>%
404    group_by(case) %>%
405    summarize(mean_by_group = mean(age))
# A tibble: 2 × 2
   case mean_by_group
  <dbl>         <dbl>
1     0          31.5
2     1          31.5
```

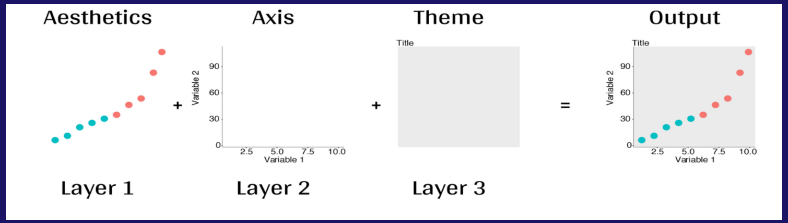
Grammar of Graphics

```
ggplot(data = <your_data_frame>, aes(<defining axes and aesthetics>)) +
  geom_<type_of_layer>() +
  labs(x = "<x_axis_label>", y = "<y_axis_label>") +
  theme(plot.title = element_text(hjust = 0.5)) +
  ggsave("<filename>.<filetype>")
```



ggplot2






```
ggplot(data = <DATA>) +
   <GEOM function>(mapping=aes(<mappings>),
                   stat = <STAT>, position = <POSITION> ) +
   <COORDINATE function> +
   <SCALE function> +
   <THEME function> +
   <FACET function> +...
```

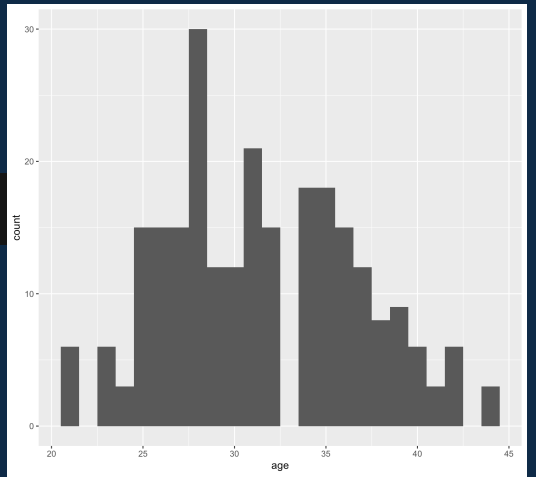
Required

Not required

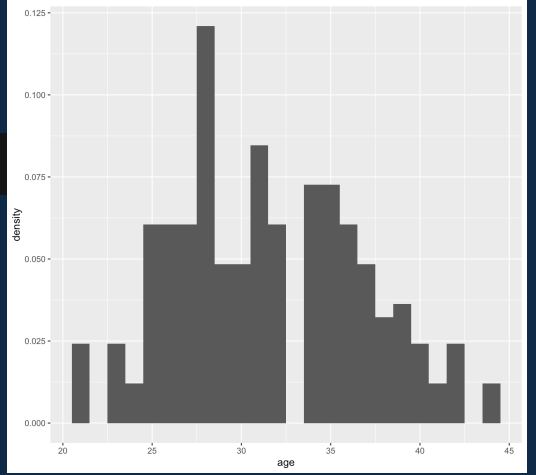
Example

ggplot

```
418  plot <- ggplot(df_s4, aes(x = age))
419  plot + geom_histogram(binwidth = 1)
```



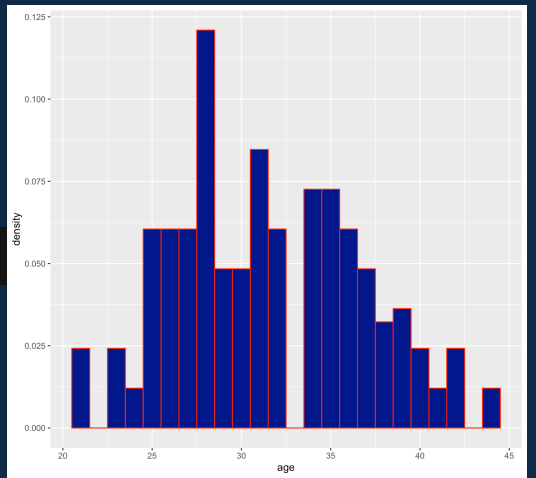

```
418  plot <- ggplot(df_s4, aes(y = after_stat(density), x = age))
419  plot + geom_histogram(binwidth = 1)
```



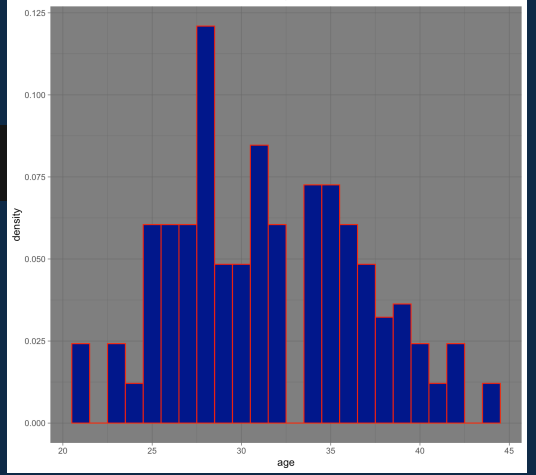
Example

ggplot

```
418 plot <- ggplot(df_s4, aes(y = after_stat(density), x = age))
419 plot + geom_histogram(binwidth = 1, color = "red", fill = "darkblue")
```



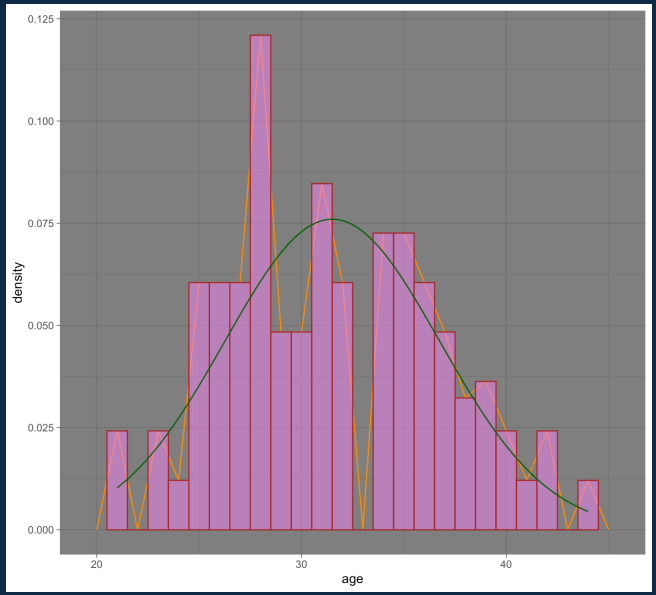

```
421 plot <- ggplot(df_s4, aes(y = after_stat(density), x = age))
422 plot + geom_histogram(binwidth = 1, color = "red", fill = "darkblue") +
423   theme_dark()
```



Example

ggplot

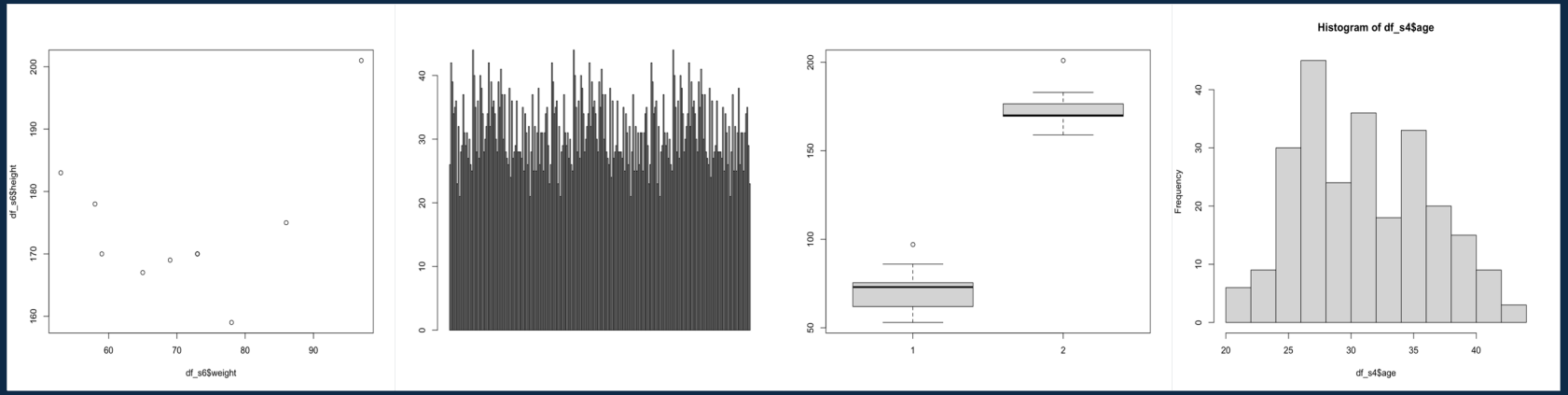
```
425 hist <- ggplot(df_s4, aes(x = age))
426 hist + geom_freqpoly(binwidth = 1, color = "darkorange", aes(y = after_stat(density))) +
427   geom_histogram(color = "brown", binwidth = 1, fill = "violet", alpha = .5, aes(y = after_stat(density))) +
428   stat_function(fun = dnorm, args = list(mean = mean(df_s4$age), sd = sd(df_s4$age)), color = "darkgreen") +
429   theme_dark()
```



Example

simple plots

```
431  plot(df_s6$weight, df_s6$height)
432  barplot(df_s4$age)
433  boxplot(df_s6$weight, df_s6$height)
434  hist(df_s4$age)
```



04

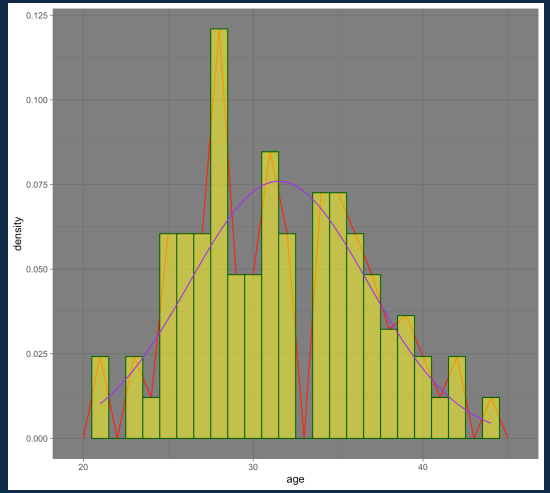
Inferential Statistics with R

Assess for Normality

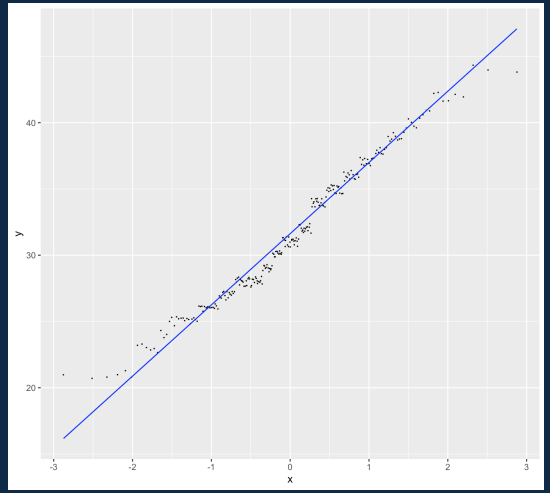
- Histogram
- Q-Q plot
- Shapiro-Wilk test → small sample size < 50
- Kolmogorov-Smirnov → larger sample size
- Anderson-Darling test → variation of K-S test, optimized for small sample size and is more powerful than S-W

Example

Histogram



Histogram with normal distribution line



Q-Q plot

Example

S-W test

K-S test

A-D test

```
452  # test for normality
453  shapiro.test(x = df_s4$age)
454  ks.test(df_s4$age, "pnorm", mean = mean(df_s4$age), sd = sd(df_s4$age))
455  ks.test(df_s4$age, "pnorm")
456  library(nortest)
457  ad.test(df_s4$age)
```

```
> # test for normality
> shapiro.test(x = df_s4$age)

        Shapiro-Wilk normality test

data:  df_s4$age
W = 0.97606, p-value = 0.0003388

> ks.test(df_s4$age, "pnorm")

        Asymptotic one-sample Kolmogorov-Smirnov test

data:  df_s4$age
D = 1, p-value < 2.2e-16
alternative hypothesis: two-sided
> ad.test(df_s4$age)

        Anderson-Darling normality test

data:  df_s4$age
A = 2.1196, p-value = 2.124e-05
```

T-test

```
> t.test(x = df_s4$age, y = df_s4$case)

        Welch Two Sample t-test

data:  df_s4$age and df_s4$case
t = 93.092, df = 251, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 30.50993 31.82878
sample estimates:
 mean of x  mean of y
31.5040323  0.3346774
```

Mann-Whitney U Test

```
> wilcox.test(x = df_s4$age, y = df_s4$case)

        Wilcoxon rank sum test with continuity correction

data:  df_s4$age and df_s4$case
W = 61504, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0
```

Correlation

```
466  cov(df_s6$weight, df_s6$height)
467  cor(df_s6$weight, df_s6$height)
468  library(psych)
469  corr.test(df_s6$weight, df_s6$height)
470  library(Hmisc)
471  rcorr(df_s6$weight, df_s6$height)
> cov(df_s6$weight, df_s6$height)
[1] 61.33333
> cor(df_s6$weight, df_s6$height)
[1] 0.3576609
> corr.test(df_s6$weight, df_s6$height)
Call:corr.test(x = df_s6$weight, y = df_s6$height)
Correlation matrix
[1] 0.36
Sample Size
[1] 9
These are the unadjusted probability values.
  The probability values  adjusted for multiple tests are in the p.adj object.
[1] 0.34

 To see confidence intervals of the correlations, print with the short=FALSE option
> rcorr(df_s6$weight, df_s6$height)
     x    y
x 1.00 0.36
y 0.36 1.00

n= 9


P
  x      y
x        0.3446
y 0.3446
```

Chi-Square

```
> chisq.test(table(df_s4$induced, df_s4$case))

        Pearson's Chi-squared test

data:  table(df_s4$induced, df_s4$case)
X-squared = 0.07323, df = 2, p-value = 0.964

> table(df_s4$induced, df_s4$case)

    0  1
  0 96 47
  1 45 23
  2 24 13
```

RESOURCES

- Codecademy Learn R
- The cartoon guide to statistics, Larry Gonick
- Discovering Statistics Using R, Andy Field
- Medical Statistics Made Easy, Gordon Taylor and Michael Harris
- Understanding Clinical Research: Behind the Statistics, University of Cape Town
- Learning R, The Hard Way Slides, Maani Beigy
- W3schools, R

THANKS !

Do you have any questions?

alirezasoltanykhaboushan@gmail.com
+98-9157707497
https://www.linkedin.com/in/alireza-soltani-709848255/
@medicosine