R

[**Books**](#_hcgvye7yrv56) **3**

[**Markdown**](#_gemixz3ri582) **3**

[**RStudio**](#_qhjl1mq0vgp6) **4**

[**Programing**](#_ep9y16fa5ol0) **4**

[Math Build-in Funcs](#_mdz2jqt09u3g) 4

[Working with Date](#_pdiphjjj8jyf) 4

[Data Types](#_5l6wimxi2kzu) 6

[Operators](#_p7yg46vmoo2a) 6

[Structures](#_rulkf1ze6l8f) 7

[…](#_sp964ar2s1ij) 8

[**DataFrame Functions**](#_m6mfnrnvjzjr) **8**

[In line dataframe](#_h9h94oksp1fd) 8

[Load Local Data](#_y172grwm3ikl) 8

[Rename Columns](#_miev2seidpvb) 8

[Remove a Column](#_bc85fhw1porj) 8

[Useful Functions](#_13ssugmf44a) 8

[Basic Data Visualization](#_g9fhyi67xnra) 9

[Create a Subset based in a Conditional](#_57csd3pp10q4) 9

[Statistical Summary of Data](#_236l7vmonysw) 9

[Family Apply](#_yau7fnpb8ri) 9

[**Packets**](#_gt6euht18p2l) **10**

[dplyr - Data Manipulation](#_plig26au7woy) 10

[tidyr - Data Reshape](#_8qv0mh9435zp) 11

[plyr - Data Split, Apply and Combine](#_2bxe1abbwdki) 12

[...](#_mhzc0c3fjaeh) 12

[**Graphics**](#_iuf3ino33xug) **12**

[**…**](#_slbjw7g25ya1) **13**

# 

*Lexical Scope* determines *where* but *not when* to look for variable values.

*R looks for variables* when the function is *executed*, not when the function is created.

# Books

R for Beginners, Emmanuel Paradis -

<https://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf>

R for Data Science, Wickham & Grolemund

https://r4ds.had.co.nz/

Advanced R, Champman & Hall’s

<https://adv-r.hadley.nz/index.html>

# Markdown

Initiate with:

---

title: "Project Title"

author: "Author Name"

date: "15 Julho, 20"

output: pdf\_document

---

```{r setup, include=FALSE}

knitr::opts\_chunk$set(echo = TRUE)

```

Follow with some text *- like jupyter lab*

## sub title

### sub sub title

Here you can write some text!

Or with some code

```{r block\_name}

# load data

df <- read.csv("credit\_dataset.csv", header = TRUE, sep = ",")

```

# RStudio

Define the new directory

setwd("<your\_path>")

Return the current directory

getwd()

Package

install.packages("packageName") - install

library(packageName) - load

detach(package:packageName) - unload

# Programing

## Math Build-in Funcs

min(set) & max(set) - return the minimum/maximum number

sqrt(numeric) - returns the square root of a number

abs(numeric) - returns the absolute value

ceiling(numeric) - rounds a numb upwards to its nearest integer

floor(numeric) - rounds a numb downwards to its nearest integer

## Working with Date

Sys.Date() - returns today’s date

date() - returns the current date and time

format(date, format=’’) - format a date

date01 - date02 - returns how much days of distance

difftime(date01, date02, units) - return in other units, likes ‘weeks’

Date type

built-in type, stored as the number of days since January 1, 1970

do not handle time, only date

as.Date(dateStr, currentFormat=’%Y-%m-%d’) - Character to Date

as.character(dates) - Date to Character

weekdays(dates) - find the day of the week

months(dates) - find the month

days(dates) - find the day

quarters(dates) - find the quarter

Package ‘chron’

handle date and time, but not time zones

store date as the number of days since January 1, 1970

store time as fractional os the day

chron( dates=dates, times=times, format=c(‘y-m-d’, ‘h:m:s’) )

*dates:* vector of str with the date

*times:* vector of str with the times

*format:* date and time format

POSIXct and POSIXlt

handle date and time with time zones

stores times to the nearest second - more accurate

*POSIXct* stores date/time as the numb of seconds since January 1, 1970

*POSIXlt* store date/time as a list with elements for second, minute, hour, day, month and year

names(DatePOSIX) - return the individual components, a data frame

From string to POSIXlt/ct

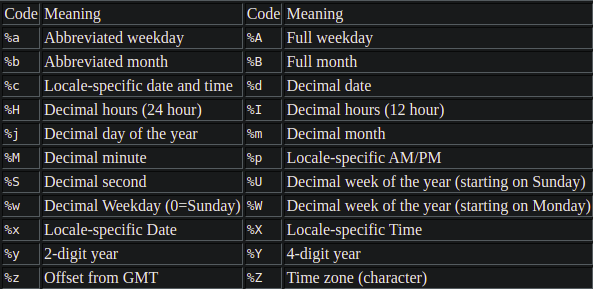
as.POSIXlt( c("2005-10-21 18:47:22","2005-10-28 07:30:05 PDT") )

From integer to POSIXlt/ct

structure( c(1127056501,1104295502), class=c(‘POSIXlt’, ‘POSIXct’) )

Inputting dates

strptime('16/Oct/2005:07:51:00', format='%d/%b/%Y:%H:%M:%S', tz=’PDT’)



## Data Types

is.type(var) - check data

as.type(var) - convert data

class(var) - return the type

**type can be:**

*array* | *vector* | *list* | *matrix*  | *logical* | *data frame*

*factor*  | *character*  | *integer*  | *numeric* | *complex*

Function dim(object) shows the object dimensions

In *integer*, it is needed to add ‘L’ after the number, i. g., 10L. There is *Inf* and *-Inf*.

A *complex* is written with an ‘i’ in the imaginary part, i. g., 3+5i

In *character*, paste( str1, str2) will merge two strings; grepl(patter, str) will search the pattern in the str - there is a family; nchar(str) will return the length; substr(str, start, stop) will extract a substring; strsplit( str, split ) will split a string in every time ‘split’ shows; str\_replace( str, pattern, replacement) will replace in a string; there is the tolower( str ) and toupper( str ).

To create a *vector*, use c() - *elements separated by comma* OR 1:10 OR seq(from, to, by).

In *vector* and *list* use LENGTH() to retrieve the size.

*List* is created using list() (can have many types) and *array* using array( seq, dim=c() ).

*Dataframe* is created using data.frame(); to show the first and last rows, use head(df) and tail(df).

Can use arithmetic operators in *array/vector/list*, *matrix* and *numeric* types.

Use matrix(c(), nrow, ncol) to create a *matrix*; diag( c() ) to create a matrix with the *c()* as its diagonal; t(matrix) for the *matrix transpose*; cbind(mat1, mat2) and rbind(mat1, mat2) will combine two matrices, by column and row; reshape() will make the data reshape.

*Array* is created using array(), each parameter *- can be a list or vector -* will be one dimension of the array, can have *n* dimensions..

Use factor(c()) to create a *factor*, categorize data. levels() show the factor levels, and can be an argument in factor(); can be ordered or not.

## Operators

Arithmetic

+ addition \* product %/% integer division

- subtraction / division ^ or \*\* exponent

%% modulus *remainder from division*

Assignment

-> or <-

->> or <<- # global assigner - used inside functions

Comparison

== equal > greater than >= greater than or equal to

!= not equal < less than <= less than or equal to

Logical

& element-wise logical AND - True if both is true

&& logical AND operator - True if both is true

| element-wise logical OR - True if one of than is true

|| logical OR operator - True is one of than is true

! logical NOT - False if statement is true

Miscellaneous

1:10 - create a series of number in a seq

%in% - find out if an element belong to a vector

%\*% - matrix multiplication

## OOP

S3 Class

Create a class using obj=list( attr1=Val1, attr2=Val2 ) OR using a constructor:

constructor <- function( attr1, attr2 ) {

attr\_list <- list(attribute1=attr1, attribute2=attr2)

attr(attr\_list, “class”) <- ‘ClassName’ #define class

attr\_list

}

obj=constructor( attr1\_val, attr2\_val )

To set a method do: print.constructor <- function(obj) {

cat(obj@attr1, “str”)

}

S4 Class

Defined using setClass( ‘ClassName’, slots=list(attr=val) ).

To create a new object, do: obj=new( ‘ClassName’, slots=list(attr=val) ).

To set a method do: setMethod(“show”, ‘ClassName’,

function(obj) {

cat(obj@attr1, “str”)

} )

## Structures

IF - ELSE

*if (cond1) {*

*statement01*

*} else if ( cond2 ) {*

*statement02*

*} else {*

*statement03*

*}*

*ifelse ( cond, returnTrue, returnFalse )*

WHILE

use break to stop the execution

use next to skip an iteration without finish

*while ( cond ) {*

*statement01*

*}*

FOR

use break to stop the execution

use next to skip an iteration without finish

*for ( i in sequence ) {*

*statement01*

*}*

Function

*my\_function <- function(param1, param2=0, …) {*

*statement01*

*return(smt\_value)*

}

## …

# DataFrame Functions

use single brackets [ ] (use index), double brackets [[ ]] or $ (last both use column name) to access columns - for multiple columns, use [[ c( ‘ColName1’, ‘ColName5’, … ) ]]

to access rows and columns, use single brackets [ , ] with index

## In line dataframe

data.frame( …, row.name=NULL)

*…:* in the form ‘value’ or ‘colName1 = c(val1, val1)’ - can have many cols

*row.name:* when the n\_row is equal to n\_col, so the function thinks the the first column is a list of column name

## Load Local Data

From a csv files

df <- read.csv("<file\_path>", sep="separator", header=T|F, row.names=NULL)

can also be ‘read.csv2()’ | ‘fread()’ | ‘read\_csv()’

*row.names:* same as above

From a xlsx files

df <- read\_excel("<file\_path>", sheet=SheetName|SheetNumb)

from package “*readxl*”

## Rename Columns

colnames(df) <- c( ‘ColName1’, ‘ColName2’, … )

## Remove a Column

df$ColName <- NULL

## Useful Functions

sort(df$col)

aggregate(col1 ~ col2, df, metric)

## Basic Data Visualization

view(df) - will open in a new tab

tail(df) - visualize the last rows

head(df) - visualize the first rows

## Create a Subset based in a Conditional

based in a conditional

conditional with the columns of the df

subset(df, cond, select=c(Column1, Column2)

## Statistical Summary of Data

str(df)

summary(df, maxsum=Int)

maxsum: numb of levels showed to factor

Factor Type: length, class and mode

Number Type: min, Q1, median, mean, Q3 and max

## Family Apply

apply(data, margin, function)

will apply a function to the rows or columns or both

*data:* matrix or data frame (input)

*margin:* 1 to rows - 2 to columns - c(1,2) to rows and columns

*function:* any build-in or custom functions

*OUTPUT:* vector, list or array

lapply(data, function)

will apply a function to all elements of the input

*data:* list, vector or data frame (input)

*function:* any build-in or custom functions

*OUTPUT:* list of the same length (use *unlist()* to transform to vector)

sapply(data, function)

will apply a function to all elements of the input

*data:* list, vector or data frame (input)

*function:* any build-in or custom functions

*OUTPUT:* vector or matrix with the same length

tapply(data, groups, function)

create a subset (with groups) and apply some func - similar to by()

*data:* object, usually a vector (input)

*groups:* a list of the groups/factors

*function:* any build-in or custom functions

* 1. …

# Packets

use df %>% action to execute two or more action in a sequential

## dplyr - Data Manipulation

*cheat sheet link*

Data Resume

glimpse(df) - show more info than str

str(df)

Create a new column

mutate(newCol = statement) - preserves existing ones

transmute(newCol = statement) - drops existing ones

Rename a column

rename(new\_name = old\_name)

Data selection

select(...) - select and rename variables in a data frame

… variables the will be select

can use *:, !, &, |, starts/ends\_with(), contains()* to select

Data filter

filter(cond) - better in ungrouped data

Grouped Data

group\_by(..., .add=F) - create grouping

ungroup(...) - removes grouping

…: variables to (un)group by

.add=T will add the group, =F will delete the existing ones

Change the ordering of the rows

arrange(..., .by\_group=F)

… variables to ordering

use desc(col) to be descending

use .by\_group=T in order to group by them

Summary Functions - Statistics

summarise(colName = statisFuncs(col))

create a col for each statistic measure (below) you want

count(col) - count numb of row in each group defined by col

n() & n\_distinct() & top\_n() -- numb of rows & numb of uniques & top n vals

mean & median & sum (col, na.rm=F) -- if na.rm = TRUE

quantile & min & max (col, na.rm=F) -- so it will drop the

IQR & mad & sd & var (col, na.rm=F) -- NA values

first(col) & last(col) & nth(col,n) - first & last & nth value

any & all (na.rm=F) - if any/all values (they’re logical) are true

Join Data Frames

*“JOIN* (df1, df2, by=”colKey”) - df1 and df2 has colKey

*Operation”* (df1, df2, by=NULL, suffix=c(“key.df1”,“key.df2”))

**Join Operation**

left\_join - right\_join - inner\_join

full\_join - semi\_join - anti\_join

union - intersect - setdiff - setequal (df1, df2)

Create a Data Sample

sample\_n(size=n)

Creates a Histogram

hist(df$col)

Check if a data belongs to an object, i. g., vector, matrix

data %in% obj

## tidyr - Data Reshape

Convert wide format to long format

gather(data, keyCol, valueCol, sourceCol, factor\_key=TRUE)

*data:* data object in wide format

*keyCol:* new key column (made from names of the data columns)

*valueCol:* new value column

*sourceCol:* data columns that contain the values use in valueCol

*factor\_key:* treat key column as a factor

*OUTPUT:* data object in long format

Convert long format to wide format

spread(data, keyCol, valueCol)

*data:* data object in long format

*keyCol:* the columns that contain the new column names

*valueCol:* the columns that contain the values

*OUTPUT:* data object in wide format

Transform a single column into multiple columns

separate(data, col, into, sep, remove=T, convert=F)

*data:* vector or df$col

*col:* column name or position

*into:* name of the new columns - c(newCol1,newCol2)

*sep:* separator between columns - can be a regular expression

*remove:* if T, remove input col from output

*convert:* if T, will run ‘type.convert()’ with ‘as.is=T’

Paste together multiple columns into one

unite(data, …, col, sep=’\_’, remove=TRUE, na.rm=FALSE)

*data:* a data frame

*…:* columns to unite - col1:coln

*col:* name of the new column

*sep:* separator to use between value

*remove:* if T, remove input col from output

*na.rm:* it T, missing values will be remove

## 

## plyr - Data Split, Apply and Combine

For data frames

ddply(data, variables, fun, …, parallel=FALSE)

*data:* the data frame

*variables* to split data frame by a formula (col1 ~ col2) or character vector (c())

*fun:* function to apply to each piece

*...:* function’s arguments

*parallel:* apply function in parallel

## ...

# Graphics

*cheat sheet link*

Basic plot

plot(x,y) - to vectors

plot(df) - to a data frame

Saving graphics

**to png**

png("Grafico1.png", width = 500, height = 500, res = 72)

plot(...)

dev.off()

**to pdf**

pdf("Grafico2.pdf")

plot(...)

dev.off()

ggPlot2

is based on the Grammar of Graphics

see cheat sheet in Drive

Examples:

**Boxplot**

boxplot(data=df, col1 ~ col2, main="Title", ylab="", xlab="", horizontal=F, col=c("blue", "red"))

**Histograma**

histograma(data, labels=T, ylim=c(0,10), breaks=10, main="Title")

**Barplot**

barplot(data, beside=F, col = c(...))

**Pie**

pie(data, labels=VAR, col = c(...))

# …