R Base Cheat Sheet

by MacaSalva via cheatography.com/93894/cs/20700/

Shortcuts

`cmd + enter` Runs the line you are in and goes to the next line

`alt + enter` Runs the line you are in

Getting help

`?` (Looks in your library)

`??` (Looks in every library)

`help.s ear ch( " _")` looks for a word or phrase

`help(p ack age ="_")` find help for a package

Getting help is very useful when you are defining functions or other things, it provides information about all the different conditions possible.

Working directory

`getwd()` Tells you your working directory

`setwd( 'C: //f ile /path')` How to code your desired wd.

You can select it manually in RStudio by clicking in the gear.

It has to be set to open correctly a .csv, if the file you desire to open is not in the same folder where you are saving your code you are going to need to define the path for opening without any problems.

Working with .csv

To open a .csv:

Whenever you open a csv, you have to save them with some name in order to be able to access to it and have it in your environment

`read.csv(path/name of the csv, ... )` When opening a csv, look at the help of the function in order to see all the needs for each case.

You can open manually a csv by clicking the import button in the environment. For this the **`re adr`** package is needed.

Result: `ds< -re ad.c sv()` its always good to open the imported dataset to see if everything is correct. `View()`

To save a .csv:

`write.c sv(df, " __",...)` When saving a .csv look into the help to see all the possible variables that can be rearrange.

Put things together

`paste( " " ,"", sep=" ")`	Pastes two things together
`paste0()`	Pastes things without separator

Creating functions

```
myfunction<- function (x){
 code
 paste()
}
```

Difference between paste, print and return

What happens inside the function stays in the function

Dates

`as.Date( __, format = " __")`

In the first blank you have to put your dates, in the format how R needs to read it.

RStudio understands that dates can exist and it knows how they work. when a variable is a date is going to show in the form :

year-month-day

we can change this when we transform it into a character. Also, and because R understands dates as said before, when you are defining it from a string you have to explain how to read it.

To get current date and time:

`Sys.Date()`

You can ask for a sequence of dates if you use `seq(as.Da te( ),a s.D ate (), by= " ")` and select if you want it to be daily, weekly, monthly etc by saying so in the by= "day"

Lists

`list(_ , _ , _ )`

To create a colum

`list[[1]]`

To access to the first element of the list

`list[[ 1]][1]`

To access to the first element of the first element of the list

To combine lists `<-c (li st1 ,list2)`

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Tables of proportions

`prop.t able(t)`	It sums up to 1
`prop.t able(t, 2)`	Column-wise
`prop.t able(t, 1)`	Row-wise

Working with NA

When working with NA's a lot of different operations don't work, because of that you have to ask for different things to obtain results

`any.NA()`

It returns TRUE if one of the elements is NA

Types of data

Logical	`TRUE, FALSE`	Boolean Values (T/F).
Numeric	`2, 5, 7`	Interger or floating point numbers.
Character	`" hel lo", " bye "`	Character strings.
Factor	`" mal e", " fem - ale "`	Character strings with different levels.
	different levels	
	levels(_)	For assessing the levels
	`NA`	Missing values

For changing for one type to other you have to use the function `as._()` and it will transform in the type you decide. You can also transform dates.

Conditions

`!`	Not
`a == b`	Are equal
`a != b`	Are not equal
`a > b`	Great than
`a < b`	Less than
`a >= b`	Greater or equal
`a <= b`	Less or equal
`is.na(a)`	Is missing
`is.null(a)`	Is null
`&`	And
`|`	Or

Operations with characters

`substr ing ("__ ", first= #, last= #)`

Returns the characters inside the string within those positions

`nchar()`

Counts the number of characters (including symbols and spaces)

Reshaping DataFrames

`melt (df, id.var s="", variab le.n am e =="")` Transforms to long

`dcast (df, id~mea sure)` Transform to wide

To ask for help look for reshape2

Look session 5

Table

`prop.t able(t)`

Returns a table with the proportions of all

`prop.t abl e(t,2)`

Proportions by colums

`prop.t abl e(t,1)`

Proportions by rows

To round numbers:

`round (x, #)`

It will round the x you ask with the decimals you ask (#)

`ceiling(#)`

It will round to the next number

`floor(#)`

It will round in the number you have

Conditionals

`if (<c ond iti on>) { <co de> }`	Just one condition
`if (<c ond iti on>) { <code> } else { <co de> }`	One condition, if not the rest without condition

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Conditionals (cont)

```
if (<c ond iti on>) {
  <co de>
} else if (<c ond iti on>
) {
   <co de>
} else {
   <co de>
}
```

More than one condition

You have to be careful when you place your conditions, the one that conditions the most have to be the first and so on

Creating Data Frames

```
df<- data.frame("name1"=c(values),
 " nam e2"= c(v alues), etc)
```

You have to define the column names and give it the values that you want. The values can be a vector, a list or other things.

For getting things from data sets: `df[[_]]` You can put the number of the row but is better putting the name.

Is very useful to use the command $ to access a data frame `df$col umname`

For adding things:

```
rbind (df1,df2)
```

Fails: row numbers don't match

```
cbind ( df1,df2)
```

Fails: column names don't match

```
aggreg ate()
```

session 6 To do aggregations and get the result formatted as a data.frame

Apply

```
lapply( __, function)
```

apply a function over a list or a vector

```
sapply(__, function)
```

same as lapply but with simplified results (better)

```
tapply(__, groupi ng, fun ction)
```

Apply a function over a ragged array

Session 6

Math

`sqrt()`	Square root of the number
`log()`	Logarithm of the number
`abs()`	Absolute value
`+`	Sum
`-`	Substraction
`*`	Multiplication
`/`	Division
`^`	Exponential
`%%`	Module operator
`union()`	Union
`inters ect( )`	Intersection
`setdiff()`	Difference
`%in%`	Membership
`pct=TRUE`	Percentage (*100)
`mean()`	Mean

Math (cont)

`median()`	Median
`sd()`	Standar deviation
`quantile()`	Quantiles
`quanti le( df$col , seq(0, 1, 0.1))`	Percentiles
`cor()`	Correlation between variables

Functions

```
summar y(df)
```

Gives you info about all the columns, (min, max, median, mean, 1n3Q)

```
head(df)
```

Gives you the first 6 lines by default, you can change it.

```
tail(df)
```

Gives you the last 6 lines by default, you can change it.

```
dim(df)
```

Gives you the dimensions of your df

```
str(df)
```

Gives you like a list with the variables

```
barplot(t)
```

Creates a barplot, not the way we are going to do them

```
length ( )
```

Returns the length

By **MacaSalva**
cheatography.com/macasalva/

Not published yet.
Last updated 8th October, 2019.

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Cheatography

R Base Cheat Sheet

by MacaSalva via cheatography.com/93894/cs/20700/

Functions (cont)

Logical operators:

`all( )`

TRUE if all elements are true

`any( )`

TRUE if any of the elements is true

To repeat things

`rep()`

`rep( _ , times=#)`

You repeat that same thing the numbers you asked

`rep( _ , each= #)`

You repeat each thing # times

`sum()`

Sum of vector elements

`seq()` or `_:_`

Generates a sequence

`cumsum()`

Cumulative sum in each position

`diff()`

Like cumsum but substracting

`nchar()`

Counts the number of characters in each position

`grep("_ ", vector ,ig nor e.c ase -=TRUE)`

Pattern Matching and Replacement -> returns position

`grepl( " _",v ect or, ign ore.ca s e= TRUE)`

Functions (cont)

Pattern Matching and Replacement-> returns logical vector

`gsub("", " ",_, ignore.ca se= FA LSE)`

For replacement, first what you want to put out, the what you want to put, and then where.

Data filtering and reordering

You can use logical conditions, they can be used in two forms:

- By creating a logical vector and applying it

```
logica l_v ector <- c(TRUE, FALSE, TRUE, FALSE )
products_stock[logical_vector]
```

-By just applying the logical condition

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

- With the function `which()` returns the positions

You can put conditions:

- Either inside the line: `df[df$col1 < 25, ]`

- Or with function subset()

```
subset( df, column name with the condition)
```

With subset you can put more than one condition with the command `&`

Reordering:

- `sort ( _,  decrea sin g=F ALSE)` by default decreasing is false and you can omit it. It rearrange the vector

- `order ( _, decrea sin g=F ALSE)` by default decreasing is false and you can omit it. It just gives the positions where they should be

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com