Cheatography

R Programming Cheat Sheet

by skydlins via cheatography.com/194596/cs/40633/

Data Structures

Vector	ordered array of elements of the same data type a<--c(3,1,5)
Vector Naming	a<-c("desks" = 1, "tables" = 3, "chairs" = 4)
Vector Coercion	a<-c(TRUE, FALSE, TRUE) = 1 0 1
	seq(1,9,2) and rep(c(2,3,4), 3)
Vector Subsetting	materials <- c(wood = 17, cloth = 36, silver = 24, gold = 3)
	materials[1] = wood = 17
Matrix	vector of elements arranged in two dimensions
	m1<-matrix(3:8,ncol=3,nrow=2)
	m2<-3:8 and dim(m2)<-c(3,2)
Factor	used to store categorical variables (numeric or character)
	a<-c(0,1,0,0,1)
	a.f<-factor(a,labels = c("Male","Female"))
	a.f = Male Female Male Male Female
gl() function	generate factors by specifying the pattern of their levels
	gl(2,8,labels=c("male","female"))
List	multiple types of elements ()list
	Mike<-list(Name="Mike",Salary=10000,Age=43,Children=c("Tom","Lily","Alice"))
#$	is a convenient way to retrieve element by element name.
str()	display the internal structure

Data Structures (cont)

c()	combine several lists into one
Array	multi-dimensional arrangement of data in a vector.

Exploring Data

Missing Data

Causes	human error, system error, loopholes
Dealing	summary() - how much data is missing
missing categorical data	set a new category called “Unknown”
missing numerical data	assign mean value or assign a value based on its relationship to other related variables
Other Data Problems	data entry, logical errors, outdated, inconsistent

Data Visualization

Principles	Simplify, Compare, Attend (Details), Explore (Visual), View diversely, Ask why, Be skeptical, Respond
GGPlot2	(+) allows us to make complex and aesthetically pleasing plots quickly and intuitively
	(-) work exclusively with data tables

Components

data	data table in the example plot is summarized.

Data Visualization (cont)

geometry	scatter plot, histograms, smooth densities, q-q plots, and blocks plots.
aesthetic mapping	x and y axis
scale	range of x-axis and y-axis appear to be defined by the range of the data
labels, title, legend,	

Creating a New Plot

ggplot() function	specify the graph’s data component.
df %>% ggplot()	associates the dataset with the plotting object
geom_point()	add a layer, assigning population to x and total to y
aes()	recognizes variables from the data component
geom_label() and geom_text()	functions to add text to the plot.
Size Color	geom_point(size = 3, color = "blue")
geom_histogram()	
geom_density()	create smooth densities

Programming Structure and Functions

Basic

if-else	use curly braces “{}
if(boolean condition){ expressions } else{ alternative expressions }	
any() (similar to OR "|")	returns TRUE if any of the logicals are true

Programming Structure and Functions (cont)

z <- c(TRUE, TRUE, FALSE) any(z)	TRUE
all() (similar to &)	returns TRUE if all of the logicals are true

Basic Functions

my_function <- function(x){ operations that operate on x which is defined by user of function value of final line is returned }	

For Loops

for (i in range of values){ operations that use i, which is changing across the range of values }	
for (i in 1:5){ print(i) }	## [1] 1 ## [1] 2 ## [1] 3 ## [1] 4 ## [1] 5
apply()	apply a function to the margin of a matrix or a dataframe
apply(x, MARGIN, FUNC, ...)	
z <- cbind(A=1:3,B=4:6,C=7:9,D=10:12)	
apply(z,2,sum)	
lapply()	works on list or vector inputs instead of matrix/dataframe input.
	returns a list of the same length as the given list or array.
x <- list(A=1:4, B=seq(0.1,1,by=0.1))	
lapply(x, mean)	
sapply()	wrapper of the lapply() function. It also takes in a list or vector, however it returns a vector instead of a list
vapply()	performs exactly like lapply() except that we can specify the return value type from FUNC

Programming Structure and Functions (cont)

	can be faster if we know that our output can use a atomic data type that takes up less memory space.
rapply()	a specified function to all elements of a list recursively
x <- list(A=2,B=list(-1,3),C=list(-2,list(-5,6)))	
rapply(x, function(x){x^2}) #returns a vector	
mapply()	take multiple vectors as inputs.
tapply()	applies the specified FUNC to each group of an array, grouped based on levels of certain factors.
Pivot Table	grouping data by different fields
	summarize the data with your own function for specific purposes
data(murders) tapply(murders$total, murders$region, sum)	
tapply(murders$total/murders$population, murders$region, mean)	
split()	split a dataframe into a list of data frames based on a factor array.
tapply()	group data by multiple factors

Basic Data Wrangling

Data Frame	use the data.frame() function. elements in the same column should be of the same data type.
	name <- c("Anne"), age <- c(28), child <- c(FALSE)
	df <- data.frame(name, age, child)
Data Frame Naming	names(df) <- c("Name", "-Age", "Child")
Data Frame Structure	Data Frame in R is implemented as a list of vectors with an important restriction of equal length vectors.
	R stores the character data type as a factor instead
str()	prevents R from converting the characters to vectors
Data Frame Subsetting	“[]” and “[[]]” and “$”
	df[3,2] #r3c2
c()	used to subset multiple portions of the Data Frame.
Data Frame Extension	adding new variables or observations to an existing Data Frame.
	height <- c(163, 177, 163, 162, 157)
	df$height <- height
Sorting	sort(df$age) #based on age

Basic Data Wrangling (cont)

	max(df$age) #getting the highest age
	which.max(df$age) #index of the oldest person
Data Frame Indexing	find specific cases in DF
	index <- df$height > 171
	sum(index) #number of people taller than the male average
	df$name[index] #person who is taller: pete
finding those older than 30 without children.	index <- df$age > 30 & df$child == FALSE
library(dplyr)	
mutate() function	extend DF for row and col
	df <- mutate(df, bmi = weight/height^2*10000)
	or df$bmi <- df$weight/df$height^2*10000
filter()	subset rows
	filter(df, bmi > 18.5 & bmi < 24.9)
select()	health <- select(df, name, height, weight, bmi)
	filter(health, bmi > 18.5 & bmi < 24.9)
%>%	chain these three functions together.
	df %>% select(name, height, weight, bmi) %>% filter(bmi > 18.5 & bmi < 24.9)

Basic Data Wrangling (cont)

merge 2 df based on col	right_join & left_join
suffix	added to the column names from each data frame to make them unique in the result.
	should be a vector with two elements
	right_join(driver_q2, constructors, by = c("constructor" = "constructor"),suffix = c("_driver", "_constructor"))
inner_join	returns only the rows that have matching values in both data frames based on specified key columns
union	combine two or more data frames vertically, stacking them on top of each other.
anti_join	filtering rows from the first data frame based on values that do not have matching values in the second data frame.
common used for df	rbind & bind_rows

Advance Data Wrangling

Importing Data	
Via readr	read_csv: comma separated values
	read_tsv: tab delimited separated values
	read_delim: general text file format
	head() function display it as a tibble.
readxl	read_excel,xls,xlsx
R-base	read.csv() and read.table() can be used without having to install any libraries
R-base import function will automatically convert any character strings to factors	
CSV	widespread use in the data science community due to its efficiency at storing large amounts of data and also as it is platform agnostic.There is also no size limit with csv files.
Via URL	read_csv(url)
tempdir() & tempfile()	it is useful to have a temporary directory or filename auto generated to manage these URL imports
Via JSON	provided via API, library(jsonlite), fromJSON(url)

Advance Data Wrangling (cont)	
Via XML	rawling a website, xmlParse("books.xml")
xmlRoot()	access the root node of the tree.
xmlChildren()	use the children nodes of the tree
xmlToList(data), xmlToDataFrame(books)	convert the XML file to list or data frame format
Reshaping Data	
Wide to Tidy: gather()	convert the above wide data into tidy data
country,year,feartility	new_tidy_data <- wide_data %>% gather(year, fertility, '1960':'2015')
Tidy to Wide: spread()	The first argument of the spread() function is to declare which variables are to be used as column names. While the second argument is to specify the variables used to fill out the cells.
Separate and Unite	
separate()	requires the target column, the names for the new columns and the separator character.
dat %>% separate(key, c("year", "first_variable_name", "second_variable_name"), fill = "right")	
spread()	

Advance Data Wrangling (cont)	
dat %>% separate(key, c("year", "variable_name"), extra = "merge") %>% spread(variable_name, value)	
unite()	first name & last name
Combining Data	
join()	combined so that matching rows are together
Inner Join	eturns only the rows that have matching values in both tables
Left Join	returns all the rows from the left table and the matching rows from the right table
Full Join	all the rows from both tables, with NULL values in columns where there is no match in the other table
Semi Join	keep the part of the first table for which we have information in the second table, but doesnt add the columns of the second.

Advance Data Wrangling (cont)	
Anti Join	opposite of the semi_join() function. It allows us to keep the part of the first table for which we have NO information in the second table, but doesnt add the columns of the second.
Set Operators	
Intersect: inds common elements shared among sets.	intersect(1:10, 6:15) = 6 7 8 9 10
Union: ombines sets into one, removing duplicates.	same with interse
Setequal	helps us check if two sets are the same regardless of order.
Setdiff	find the elements that are in one set (or vector) but not in another set.