PSYC 259: Principles of Data Science

Week 4: Part 2
Automation

Automation

What to automate?

- Replacing manual copy/paste/renaming with data-cleaning scripts
- Replacing drop-down menu analyses with scripts

What to automate?

- Replacing manual copy/paste/renaming with data-cleaning scripts
- Replacing drop-down menu analyses with scripts
- *Replacing redundant code with more efficient code*

Types of code to avoid writing

- Hard coding
  - ds[1, 1]
  - proportion <- ds$counts / 5365

Types of code to avoid writing

- Hard coding
- Repetitive coding
  - summarize(m_a = mean(a), m_b = mean(b), m_c = mean(c), m_d = mean(d))
  - ds_a <- read_csv("data_a.csv")
  - ds_b <- read_csv("data_b.csv")
  - ds_c <- read_csv("data_b.csv")

  - Copy-paste-tweak leads to coding mistakes!

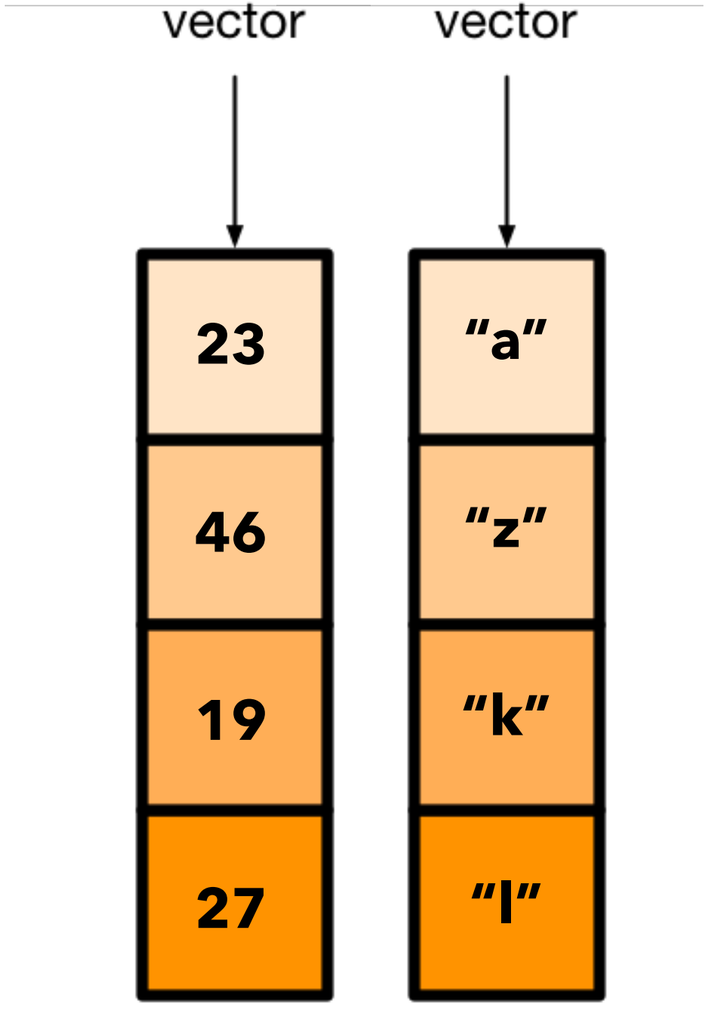
Tools to avoid repetitive coding

- select(col1:col50)
- rename_with(make_clean_names)
- mutate(across(selected_vars), list(fxs))
- summarize(across(selected_vars), list(fxs))
- vroom(list_of_files)

- What do they all have in common?

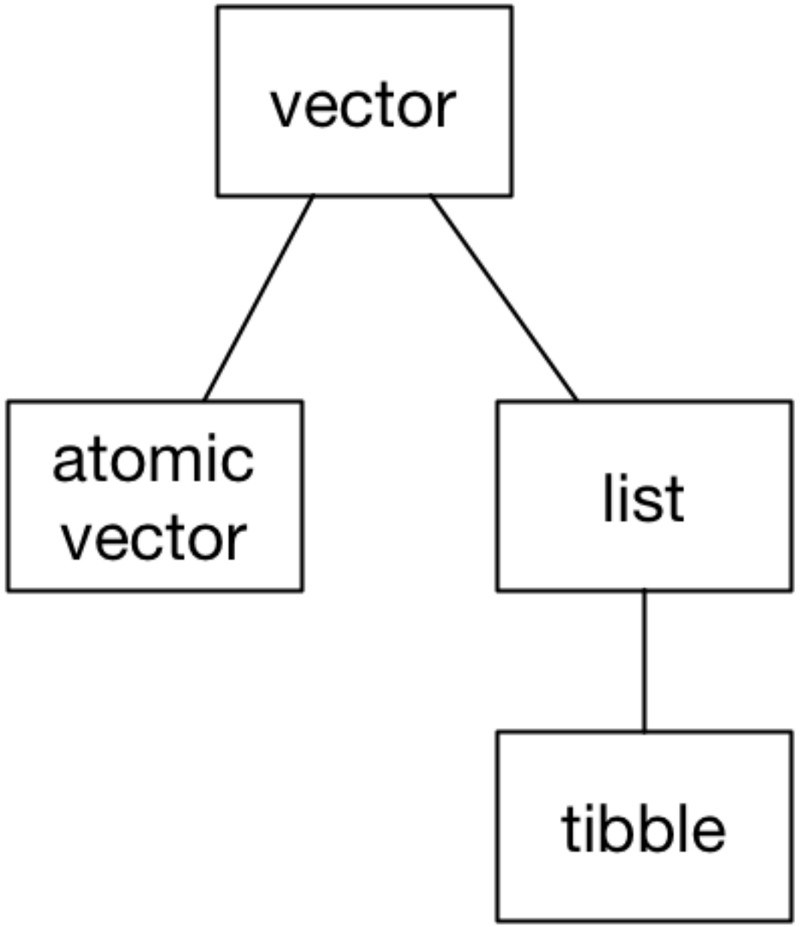
They all operate across **vectors**

- Vectors are collections of values
  - *atomic vectors* contain only a single type
  - numeric vector
  - logical vector
  - character vector
- v[i] accesses an element by position
  - v[1] = 23
- create vectors with v <- c(23, 46, 19, 27)



Vectors vs. lists vs. tibbles

- List: a mixed-type vector
  - x <- list("c", 1, TRUE)
  - Lists can contain lists, whereas vectors are unnested
- Tibbles are lists of vectors
  - All vectors are the same length
  - Each vector has a different type



Named elements help us get away from hard coding

No names
Access by position only

```
> x <- c("X", "Y", "Z")
> x
[1] "X" "Y" "Z"
> x[1]
[1] "X"
```

With names
Access by position or name

```
> y <- c(first = "X", second = "Y", third = "Z")
> y
 first second  third
   "X"    "Y"    "Z"
> y["second"]
second
   "Y"
> names(y)
[1] "first"  "second" "third"
> set_names(y, c("a", "b", "c"))
  a   b   c
"X" "Y" "Z"
```

Iterating over vectors/lists

Three main options to iterate/loop

- Loop -> perform a set of actions for each element of a vector/list
  - for loops
  - map (tidyverse *purrr* package)
  - lapply (base R, like map but harder to use)

For loops

Var

It takes items from iterable one by one

Iterable

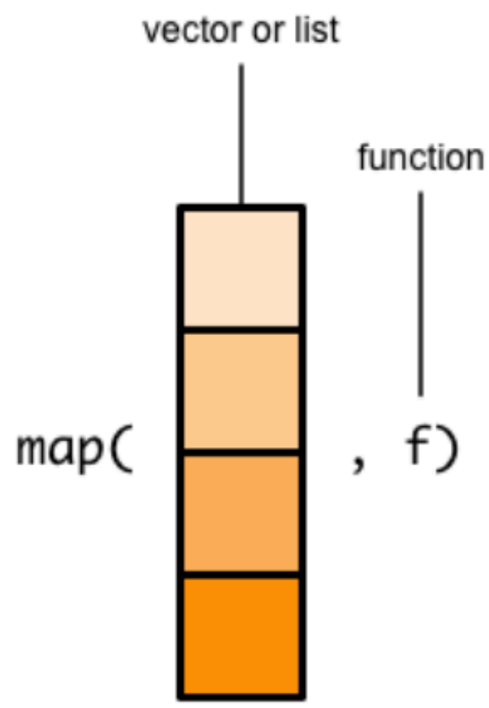
It's a collection of objects (like a vector, list etc.)

```
for (var in iterable) {
    statement
    statement
        ...
}
following_statement
```

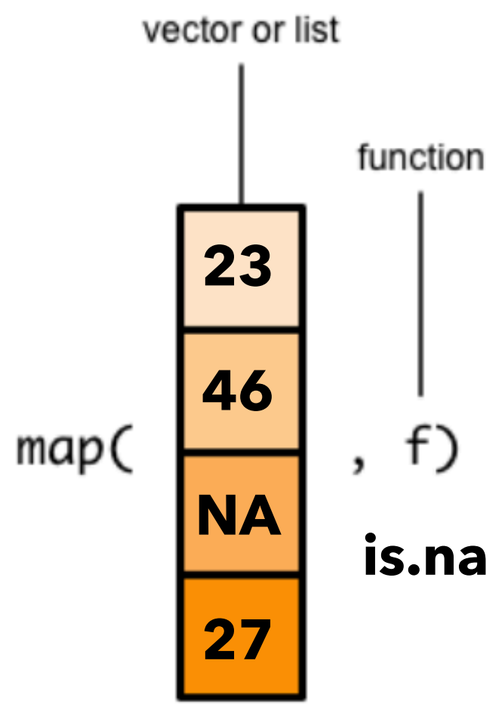
Loop body

It is executed once for each item in iterable

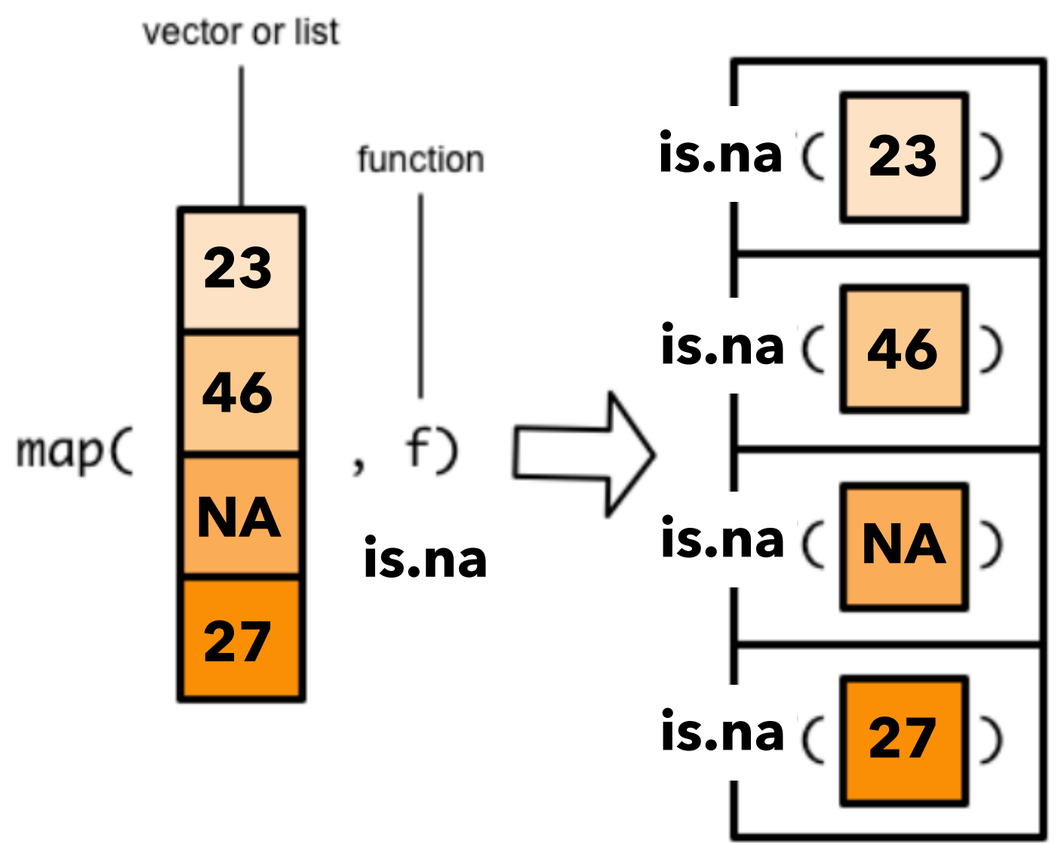
map



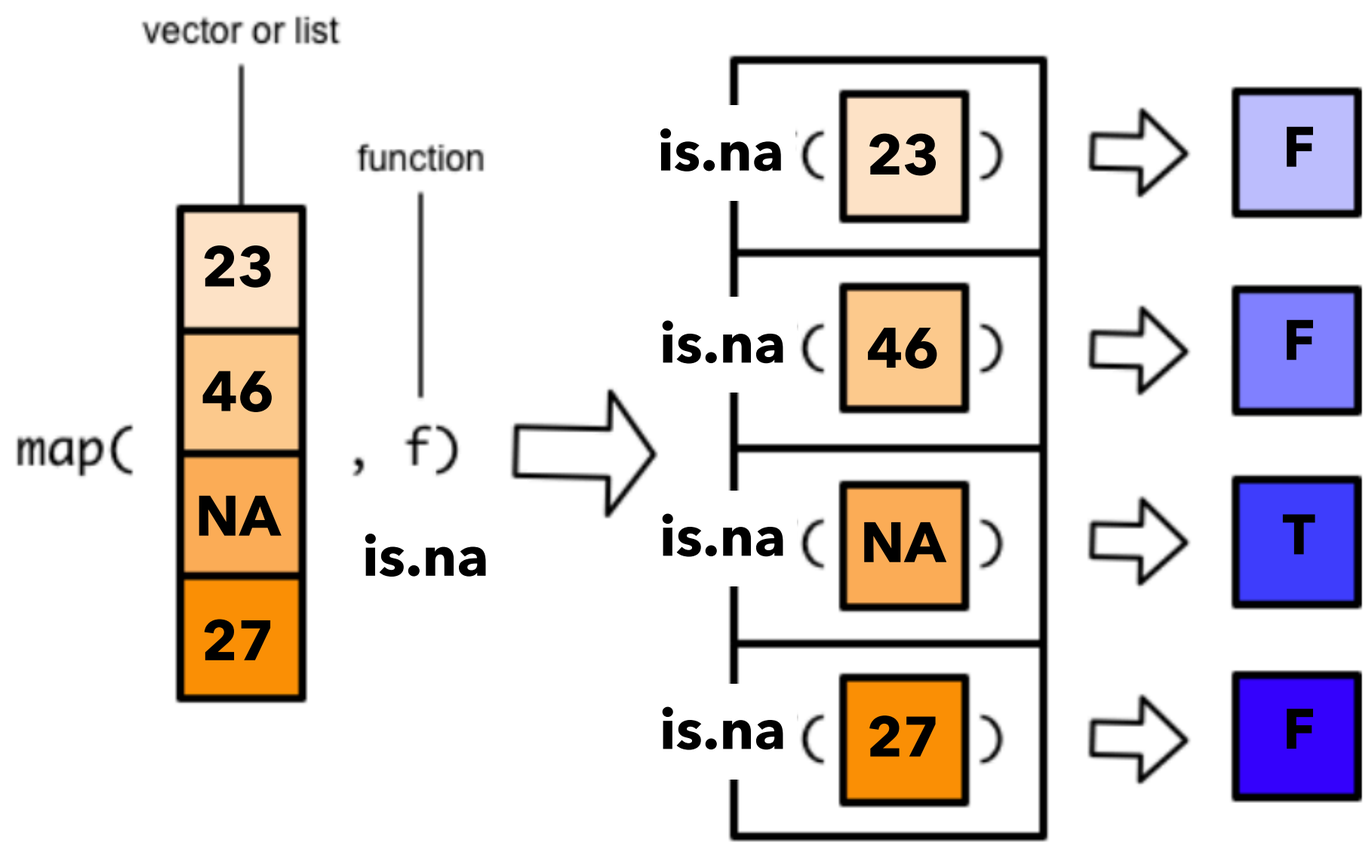
map



map


vector or list
function
23
46
NA
27
map(
, f)
is.na
is.na (23)
is.na (46)
is.na (NA)
is.na (27)

map


vector or list
function
map(
23
46
NA
27
, f)
is.na
is.na (23)
is.na (46)
is.na (NA)
is.na (27)
F
F
T
F

Different map outputs

- map returns a list

```
> v <- c(23, 46, NA, 27)
> map(v, is.na)
[[1]]
[1] FALSE

[[2]]
[1] FALSE

[[3]]
[1] TRUE

[[4]]
[1] FALSE
```

Different map outputs

- map returns a list
- map_lgl returns a vector of logicals
- map_chr returns a vector of characters
- vector/list names get carried through

```
> v <- c(23, 46, NA, 27)
> map(v, is.na)
[[1]]
[1] FALSE

[[2]]
[1] FALSE

[[3]]
[1] TRUE

[[4]]
[1] FALSE
```

```
> v <- c(23, 46, NA, 27)
> map_lgl(v, is.na)
[1] FALSE FALSE  TRUE FALSE
> map_chr(v, is.na)
[1] "FALSE" "FALSE" "TRUE"  "FALSE"
```

```
> v <- c(w = 23, x = 46, y = NA, z = 27)
> map_chr(v, is.na)
      w       x       y       z
"FALSE" "FALSE"  "TRUE" "FALSE"
```

Other considerations

- Write code from the inside out
  - Test the basic function(s) with a single unit of data to make sure it works
  - Then, revise it to iterate over a list/vector
- Writing custom functions can make map easier (more on this next week)

Automation tutorial

“259-automation”

Homework

now on GitHub

PSYC-259-Data-Science / 259-tidying-automation-homework

Unwatch 1 | Star 0 | Fork 0

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

main | 1 branch | 0 tags | Go to file | Add file | Code

JohnFranchak added homework and instructions 52f5c5b 3 hours ago 2 commits

.gitattributes	Initial commit	3 hours ago
.gitignore	Initial commit	3 hours ago
README.md	added homework and instructions	3 hours ago
tidying_automation_homework.R	added homework and instructions	3 hours ago

README.md

259-tidying-automation-homework

For this homework assignment, you will need to use git/GitHub. First, you should fork this repository to create your own copy. Clone your copy to work on it on your own local computer or to RStudio Cloud using the instructions we went over in class.

As you work on the homework, make commits and push your changes to GitHub. When you are done with the assignment, make sure that your repository is public and submit the link to your GitHub repository on Canvas in the textbook (where you had been writing "Finished").

Since this is the first assignment where you are creating your own RStudio project (rather than working in one I created on RStudio Cloud), I included the line "install.packages("tidyverse")" so that you can install the tidyverse package. If it's already installed, you can delete or comment out that line. No other packages are needed for this homework.

About

No description, website, or topics provided.

Readme

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

R 100.0%

Homework 3 - Tidying and Automation

Due Feb 3 by 12:30pm **Points** 1 **Submitting** a website url

Unlike homeworks 1 and 2 (which were on RStudio Cloud), homework 3 will be accessed on GitHub here: https://github.com/PSYC-259-Data-Science/259-tidying-automation-homework

Follow the directions to make your own GitHub repository. You should commit and push your completed assignment to your own public GitHub repo. When you are finished, come back to Canvas and submit the link to your GitHub repository to submit the assignment.

Website URL | Office 365

Copy and paste the link to the web site you'd like to submit for this assignment.

Website URL:

Comments...

Cancel | Submit Assignment