DSApps 2023 @ TAU: Assignment 2

The Tidyverse - Part B

Giora Simchoni

2023-04-23

Table of Contents

### Welcome

Welcome to Assignment 2 in R!

Remember:

* You can play with the assignment in Playground mode, but:
* Only your private Github repository assigned to you by the course admin will be cloned and graded (Submission mode, see instructions [here](https://github.com/DSApps-2023/Class_Slides/blob/master/Apps_of_DS_HW.pdf))
* Like any other University assignment, your work should remain private
* You need to git clone your private Github repository locally as explained [here](https://github.com/DSApps-2023/Class_Slides/blob/master/Apps_of_DS_HW.pdf)
* You need to uncomment the starter code inside the chunk, replace the ### YOUR CODE HERE ###, run the chunk and see that you’re getting the expected result
* Pay attention to what you’re asked to do and the required output
* For example, using a *different* function than the one you were specifically asked to use, will decrease your score (unless you amaze me)
* Your notebook should run smoothly from start to end if someone presses in the RStudio toolbar Run –> Restart R and Run All Chunks
* When you’re done knit the entire notebook into a html file, this is the file that would be graded
* You can add other files but do not delete any files
* Commit your work and push to your private Github repository as explained [here](https://github.com/DSApps-2023/Class_Slides/blob/master/Apps_of_DS_HW.pdf)

This assignment is due: 26/4 23:59

### Packages

These are the packages you will need. If you don’t have them, you need to uncomment the install.packages() line and install them first (you can also just copy this command to the R console and do it there if you don’t want all the output printed in this notebook).

When you load the packages you may see different kinds of messages or warnings, skim them:

# install.packages(c("tidyverse", "repurrrsive")) # readxl should be installed with tidyverse  
library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.1 ✔ readr 2.1.4  
## ✔ forcats 1.0.0 ✔ stringr 1.5.0  
## ✔ ggplot2 3.4.1 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.2 ✔ tidyr 1.3.0  
## ✔ purrr 1.0.1   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ℹ Use the ]8;;http://conflicted.r-lib.org/conflicted package]8;; to force all conflicts to become errors

library(readxl)  
library(repurrrsive)

### Tidying Data

##### (40 points)

Go over the [UN Migration Dataset](https://DSApps-2023.github.io/Class_Slides/u1_d03-wranglingB/u1_d03-wranglingB.html#25) we’ve talked about in class.

Open the Excel file in this repo’s data folder and see for yourself how untidy an official dataset can get. If you don’t have Excel look again at the [screenshot](https://DSApps-2023.github.io/Class_Slides/u1_d03-wranglingB/u1_d03-wranglingB.html#26) from class.

In this part of the assignment you will recreate the steps needed to get to the (almost) tidy migration dataset you can find in the [class repo](https://github.com/DSApps-2023/Class_Slides/tree/master/data).

First read in the 3rd sheet of this Excel file (a.k.a “Table 2. Male migrant stock at mid-year…”), with the [read\_excel()](https://readxl.tidyverse.org/reference/read_excel.html) function from the readxl package:

dirty\_data <- read\_excel("data/UN\_MigrantStockByOriginAndDestination\_2019.xlsx", sheet = 3)

## New names:  
## • `` -> `...1`  
## • `` -> `...2`  
## • `` -> `...3`  
## • `` -> `...4`  
## • `` -> `...6`  
## • `` -> `...7`  
## • `` -> `...8`  
## • `` -> `...9`  
## • `` -> `...10`  
## • `` -> `...11`  
## • `` -> `...12`  
## • `` -> `...13`  
## • `` -> `...14`  
## • `` -> `...15`  
## • `` -> `...16`  
## • `` -> `...17`  
## • `` -> `...18`  
## • `` -> `...19`  
## • `` -> `...20`  
## • `` -> `...21`  
## • `` -> `...22`  
## • `` -> `...23`  
## • `` -> `...24`  
## • `` -> `...25`  
## • `` -> `...26`  
## • `` -> `...27`  
## • `` -> `...28`  
## • `` -> `...29`  
## • `` -> `...30`  
## • `` -> `...31`  
## • `` -> `...32`  
## • `` -> `...33`  
## • `` -> `...34`  
## • `` -> `...35`  
## • `` -> `...36`  
## • `` -> `...37`  
## • `` -> `...38`  
## • `` -> `...39`  
## • `` -> `...40`  
## • `` -> `...41`  
## • `` -> `...42`  
## • `` -> `...43`  
## • `` -> `...44`  
## • `` -> `...45`  
## • `` -> `...46`  
## • `` -> `...47`  
## • `` -> `...48`  
## • `` -> `...49`  
## • `` -> `...50`  
## • `` -> `...51`  
## • `` -> `...52`  
## • `` -> `...53`  
## • `` -> `...54`  
## • `` -> `...55`  
## • `` -> `...56`  
## • `` -> `...57`  
## • `` -> `...58`  
## • `` -> `...59`  
## • `` -> `...60`  
## • `` -> `...61`  
## • `` -> `...62`  
## • `` -> `...63`  
## • `` -> `...64`  
## • `` -> `...65`  
## • `` -> `...66`  
## • `` -> `...67`  
## • `` -> `...68`  
## • `` -> `...69`  
## • `` -> `...70`  
## • `` -> `...71`  
## • `` -> `...72`  
## • `` -> `...73`  
## • `` -> `...74`  
## • `` -> `...75`  
## • `` -> `...76`  
## • `` -> `...77`  
## • `` -> `...78`  
## • `` -> `...79`  
## • `` -> `...80`  
## • `` -> `...81`  
## • `` -> `...82`  
## • `` -> `...83`  
## • `` -> `...84`  
## • `` -> `...85`  
## • `` -> `...86`  
## • `` -> `...87`  
## • `` -> `...88`  
## • `` -> `...89`  
## • `` -> `...90`  
## • `` -> `...91`  
## • `` -> `...92`  
## • `` -> `...93`  
## • `` -> `...94`  
## • `` -> `...95`  
## • `` -> `...96`  
## • `` -> `...97`  
## • `` -> `...98`  
## • `` -> `...99`  
## • `` -> `...100`  
## • `` -> `...101`  
## • `` -> `...102`  
## • `` -> `...103`  
## • `` -> `...104`  
## • `` -> `...105`  
## • `` -> `...106`  
## • `` -> `...107`  
## • `` -> `...108`  
## • `` -> `...109`  
## • `` -> `...110`  
## • `` -> `...111`  
## • `` -> `...112`  
## • `` -> `...113`  
## • `` -> `...114`  
## • `` -> `...115`  
## • `` -> `...116`  
## • `` -> `...117`  
## • `` -> `...118`  
## • `` -> `...119`  
## • `` -> `...120`  
## • `` -> `...121`  
## • `` -> `...122`  
## • `` -> `...123`  
## • `` -> `...124`  
## • `` -> `...125`  
## • `` -> `...126`  
## • `` -> `...127`  
## • `` -> `...128`  
## • `` -> `...129`  
## • `` -> `...130`  
## • `` -> `...131`  
## • `` -> `...132`  
## • `` -> `...133`  
## • `` -> `...134`  
## • `` -> `...135`  
## • `` -> `...136`  
## • `` -> `...137`  
## • `` -> `...138`  
## • `` -> `...139`  
## • `` -> `...140`  
## • `` -> `...141`  
## • `` -> `...142`  
## • `` -> `...143`  
## • `` -> `...144`  
## • `` -> `...145`  
## • `` -> `...146`  
## • `` -> `...147`  
## • `` -> `...148`  
## • `` -> `...149`  
## • `` -> `...150`  
## • `` -> `...151`  
## • `` -> `...152`  
## • `` -> `...153`  
## • `` -> `...154`  
## • `` -> `...155`  
## • `` -> `...156`  
## • `` -> `...157`  
## • `` -> `...158`  
## • `` -> `...159`  
## • `` -> `...160`  
## • `` -> `...161`  
## • `` -> `...162`  
## • `` -> `...163`  
## • `` -> `...164`  
## • `` -> `...165`  
## • `` -> `...166`  
## • `` -> `...167`  
## • `` -> `...168`  
## • `` -> `...169`  
## • `` -> `...170`  
## • `` -> `...171`  
## • `` -> `...172`  
## • `` -> `...173`  
## • `` -> `...174`  
## • `` -> `...175`  
## • `` -> `...176`  
## • `` -> `...177`  
## • `` -> `...178`  
## • `` -> `...179`  
## • `` -> `...180`  
## • `` -> `...181`  
## • `` -> `...182`  
## • `` -> `...183`  
## • `` -> `...184`  
## • `` -> `...185`  
## • `` -> `...186`  
## • `` -> `...187`  
## • `` -> `...188`  
## • `` -> `...189`  
## • `` -> `...190`  
## • `` -> `...191`  
## • `` -> `...192`  
## • `` -> `...193`  
## • `` -> `...194`  
## • `` -> `...195`  
## • `` -> `...196`  
## • `` -> `...197`  
## • `` -> `...198`  
## • `` -> `...199`  
## • `` -> `...200`  
## • `` -> `...201`  
## • `` -> `...202`  
## • `` -> `...203`  
## • `` -> `...204`  
## • `` -> `...205`  
## • `` -> `...206`  
## • `` -> `...207`  
## • `` -> `...208`  
## • `` -> `...209`  
## • `` -> `...210`  
## • `` -> `...211`  
## • `` -> `...212`  
## • `` -> `...213`  
## • `` -> `...214`  
## • `` -> `...215`  
## • `` -> `...216`  
## • `` -> `...217`  
## • `` -> `...218`  
## • `` -> `...219`  
## • `` -> `...220`  
## • `` -> `...221`  
## • `` -> `...222`  
## • `` -> `...223`  
## • `` -> `...224`  
## • `` -> `...225`  
## • `` -> `...226`  
## • `` -> `...227`  
## • `` -> `...228`  
## • `` -> `...229`  
## • `` -> `...230`  
## • `` -> `...231`  
## • `` -> `...232`  
## • `` -> `...233`  
## • `` -> `...234`  
## • `` -> `...235`  
## • `` -> `...236`  
## • `` -> `...237`  
## • `` -> `...238`  
## • `` -> `...239`  
## • `` -> `...240`  
## • `` -> `...241`

If you look at what you got (e.g. with View(dirty\_data) in RStudio) you would be appalled. Hopefully you would then reach the conclusion you do not need the first 15 rows. Find out how to read the sheet **again** skipping the first 15 rows.

#view(dirty\_data)  
dirty\_data <- read\_excel("data/UN\_MigrantStockByOriginAndDestination\_2019.xlsx", sheet = 3, skip = 15)

## New names:  
## • `` -> `...1`  
## • `` -> `...2`  
## • `` -> `...3`  
## • `` -> `...4`  
## • `` -> `...5`  
## • `` -> `...6`

The data looks much better, but still. R missed the names of the first few columns (because they appear in merged cells!) so we need to change them:

colnames(dirty\_data)[1:6] <- c("year", "order", "country", "notes", "code", "type\_of\_data")

Next look at some of the countries column names:

colnames(dirty\_data)[c(13, 17, 34, 61)]

## [1] "American Samoa" "Antigua and Barbuda"   
## [3] "Bolivia (Plurinational State of)" "Côte d'Ivoire"

These are definitely not acceptable R column names, we see spaces, parentheses, French letters, and other issues. Usually you might be able to use the make\_clean\_names() function from the [janitor](http://sfirke.github.io/janitor/) package, but for out-of-scope reasons it won’t work here.

Read the clean\_country\_name() function and use it to clean the column names of the data:

clean\_country\_name <- function(name) {  
 str\_replace\_all(str\_replace\_all(str\_to\_lower(name), " |-", "\_"), "'|,|\\(|\\)", "")  
}  
  
colnames(dirty\_data) <- clean\_country\_name(colnames(dirty\_data))

You should see the problematic column names improved (though not perfect):

colnames(dirty\_data)[c(13, 17, 34, 61)]

## [1] "american\_samoa" "antigua\_and\_barbuda"   
## [3] "bolivia\_plurinational\_state\_of" "côte\_divoire"

Another issue with this dataset is that many of the rows are simply subtotals of the number of immigrants per specific area, e.g. row 1 (17 in the original Excel sheet) is the total for the entire world. Luckily there is a way to exclude all such rows, with the code column, filter only rows with code smaller than 900:

dirty\_data <- dirty\_data %>% filter(code < 900)

While you’re at it, we don’t need columns c(4, 6, 7, 8, 9):

dirty\_data <- dirty\_data[,-c(4, 6, 7, 8, 9)]

Check you have a table with 1624 rows and 236 columns.

dim(dirty\_data)

## [1] 1624 236

Now let’s use the clean\_country\_name() to also clean the country column:

dirty\_data <- dirty\_data %>%  
 mutate(country = clean\_country\_name(country))

There are different values for missing values as said in class and we are going to assume NA means 0 migration:

dirty\_data <- dirty\_data %>%  
 mutate(across(where(is.character), ~na\_if(., "-"))) %>%  
 mutate(across(where(is.character), ~na\_if(., ".."))) %>%  
 replace(is.na(.), "0")

Finally, you may be shocked to know that all country columns (all but the first 4 columns) which contain the **number** of migrants from the row country to the column country, are actually character columns! Convert them as.numeric:

dirty\_data <- dirty\_data %>%  
 mutate(across(-c("year","order","country","code"), as.numeric))

A big problem is that dirty\_data now holds only the data for men. We can’t just copy-paste everything, we need a function. Moreover, many of the steps could be united into a single pipe.

Complete the tidy\_up\_data() function, which receives a .sheet number and a .gender string, and returns the (almost) tidy data for that gender. Uncomment the usage of the function to get the migration\_men and migration\_women datasets. Bonus points if you can wrap clean\_country\_name() inside a function called clean\_country\_name\_data() to clean the names of the dataset **inside a pipe**, not as done above. Also notice there is no country\_dest column, I want you to rename the country column.

tidy\_up\_data <- function(.sheet, .gender) {  
 dirty\_data <- read\_excel("data/UN\_MigrantStockByOriginAndDestination\_2019.xlsx", sheet = .sheet, skip = 15)  
 colnames(dirty\_data)[1:6] <- c("year", "order", "country", "notes", "code", "type\_of\_data")  
 dirty\_data %>%  
 rename(country\_dest = country) %>%  
 mutate(gender = .gender) %>%  
 select(gender, year, code, country\_dest, everything())  
}  
  
# Usage  
migration\_men <- tidy\_up\_data(3, "men")

## New names:  
## • `` -> `...1`  
## • `` -> `...2`  
## • `` -> `...3`  
## • `` -> `...4`  
## • `` -> `...5`  
## • `` -> `...6`

migration\_women <- tidy\_up\_data(4, "women")

## New names:  
## • `` -> `...1`  
## • `` -> `...2`  
## • `` -> `...3`  
## • `` -> `...4`  
## • `` -> `...5`  
## • `` -> `...6`

Bind the two datasets with the bind\_rows() function and you have the migration dataset we used in class. You can save it as a R object with the write\_rds() function:

migration <- bind\_rows(migration\_men, migration\_women)  
  
write\_rds(migration, "data/migration.rds")

Be sure to use pivot\_longer() and pivot\_wider() as we did in class to go back and forth between wide and long formats:

migration\_long <- migration %>%  
 pivot\_longer(cols = -c(1:5),  
 names\_to = "country\_orig",  
 values\_to = "n\_migrants")  
migration\_long

## # A tibble: 938,994 × 7  
## gender year code country\_dest order country\_orig n\_migrants  
## <chr> <dbl> <dbl> <chr> <chr> <chr> <chr>   
## 1 men 1990 900 WORLD 1990001 notes <NA>   
## 2 men 1990 900 WORLD 1990001 type\_of\_data <NA>   
## 3 men 1990 900 WORLD 1990001 Total 77661689   
## 4 men 1990 900 WORLD 1990001 Other South 3412163   
## 5 men 1990 900 WORLD 1990001 Other North 1159981   
## 6 men 1990 900 WORLD 1990001 Afghanistan 3711922   
## 7 men 1990 900 WORLD 1990001 Albania 109462   
## 8 men 1990 900 WORLD 1990001 Algeria 521333   
## 9 men 1990 900 WORLD 1990001 American Samoa 1049   
## 10 men 1990 900 WORLD 1990001 Andorra 1769   
## # ℹ 938,984 more rows

migration\_wide <- migration\_long %>%  
 pivot\_wider(id\_cols = 1:5,  
 names\_from = country\_orig,  
 values\_from = n\_migrants)  
migration\_wide

## # A tibble: 3,962 × 242  
## gender year code country\_dest order notes type\_of\_data Total `Other South`  
## <chr> <dbl> <dbl> <chr> <chr> <chr> <chr> <chr> <chr>   
## 1 men 1990 900 WORLD 1990… <NA> <NA> 7766… 3412163   
## 2 men 1990 NA UN developme… 1990… <NA> <NA> .. ..   
## 3 men 1990 901 More develop… 1990… b <NA> 4042… 1809849   
## 4 men 1990 902 Less develop… 1990… c <NA> 3723… 1602314   
## 5 men 1990 941 Least develo… 1990… d <NA> 5550… 244501   
## 6 men 1990 934 Less develop… 1990… <NA> <NA> 3168… 1357813   
## 7 men 1990 NA World Bank i… 1990… <NA> <NA> .. ..   
## 8 men 1990 1503 High-income … 1990… e <NA> 3999… 2034941   
## 9 men 1990 1517 Middle-incom… 1990… e <NA> 3263… 942277   
## 10 men 1990 1502 Upper-middle… 1990… e <NA> 1687… 476665   
## # ℹ 3,952 more rows  
## # ℹ 233 more variables: `Other North` <chr>, Afghanistan <chr>, Albania <chr>,  
## # Algeria <chr>, `American Samoa` <chr>, Andorra <chr>, Angola <chr>,  
## # Anguilla <chr>, `Antigua and Barbuda` <chr>, Argentina <chr>,  
## # Armenia <chr>, Aruba <chr>, Australia <chr>, Austria <chr>,  
## # Azerbaijan <chr>, Bahamas <chr>, Bahrain <chr>, Bangladesh <chr>,  
## # Barbados <chr>, Belarus <chr>, Belgium <chr>, Belize <chr>, Benin <chr>, …

### Iterating with purrr - intro

##### (0 points)

Remember the Starwars datasets from class?

sw\_tables <- read\_rds("data/sw\_tables.rds")  
characters <- sw\_tables$characters  
planets <- sw\_tables$planets  
films <- sw\_tables$films  
  
# For example the characters table:  
characters %>%  
 select(character\_id, name, gender, hair\_color, film\_id) %>%  
 head(7)

## # A tibble: 7 × 5  
## character\_id name gender hair\_color film\_id  
## <int> <chr> <chr> <chr> <dbl>  
## 1 1 Luke Skywalker male blond 1  
## 2 1 Luke Skywalker male blond 2  
## 3 1 Luke Skywalker male blond 3  
## 4 1 Luke Skywalker male blond 6  
## 5 1 Luke Skywalker male blond 7  
## 6 2 C-3PO <NA> <NA> 1  
## 7 2 C-3PO <NA> <NA> 2

Yes, well, the sad truth is that often data do not come in such a ready-made, easy-to-use, way. The Starwars datasets actually come from [SWAPI](https://swapi.co/), which is an API which lets you query the Starwars dataset, located at a distant server, and get results in JSON format, which is basically a nested list (a list of lists of lists etc.). Feel free to explore it online, have a look at other APIs like the [Chuck Norris Jokes](https://api.chucknorris.io/) API.

The raw Starwars lists are found in the repurrrsive package:

* sw\_people
* sw\_films
* sw\_planets
* sw\_species
* sw\_vehicles
* sw\_starships

For example to explore the sw\_people list interactively in RStudio:

# Do not change the "eval=FALSE" chunk option!  
View(sw\_people)



Each element of the list is a SW character, and in itself is a list of *varying* length of attributes.

For example, Luke Skywalker has the attribute “gender”:

sw\_people[[1]][c("name", "gender")]

## $name  
## [1] "Luke Skywalker"  
##   
## $gender  
## [1] "male"

C-3PO has this attribute but it is marked with a string “n/a”:

sw\_people[[2]][c("name", "gender")]

## $name  
## [1] "C-3PO"  
##   
## $gender  
## [1] "n/a"

Luke also has an attribute called “vehicles” which in itself is a list of vehicles “ID”s:

sw\_people[[1]]$vehicles

## [1] "http://swapi.co/api/vehicles/14/" "http://swapi.co/api/vehicles/30/"

But C-3PO does not:

sw\_people[[2]]$vehicles

## NULL

Notice in R this does not return an error, but NULL.

Now. Let us use purrr to reconstruct a **table** out of this nested list, similar to what l used in class.

Getting all SW characters names from sw\_people can easily be done with map if you want a list or map\_chr if you want a vector:

map\_chr(sw\_people, "name")

## [1] "Luke Skywalker" "C-3PO" "R2-D2"   
## [4] "Darth Vader" "Leia Organa" "Owen Lars"   
## [7] "Beru Whitesun lars" "R5-D4" "Biggs Darklighter"   
## [10] "Obi-Wan Kenobi" "Anakin Skywalker" "Wilhuff Tarkin"   
## [13] "Chewbacca" "Han Solo" "Greedo"   
## [16] "Jabba Desilijic Tiure" "Wedge Antilles" "Jek Tono Porkins"   
## [19] "Yoda" "Palpatine" "Boba Fett"   
## [22] "IG-88" "Bossk" "Lando Calrissian"   
## [25] "Lobot" "Ackbar" "Mon Mothma"   
## [28] "Arvel Crynyd" "Wicket Systri Warrick" "Nien Nunb"   
## [31] "Qui-Gon Jinn" "Nute Gunray" "Finis Valorum"   
## [34] "Jar Jar Binks" "Roos Tarpals" "Rugor Nass"   
## [37] "Ric Olié" "Watto" "Sebulba"   
## [40] "Quarsh Panaka" "Shmi Skywalker" "Darth Maul"   
## [43] "Bib Fortuna" "Ayla Secura" "Dud Bolt"   
## [46] "Gasgano" "Ben Quadinaros" "Mace Windu"   
## [49] "Ki-Adi-Mundi" "Kit Fisto" "Eeth Koth"   
## [52] "Adi Gallia" "Saesee Tiin" "Yarael Poof"   
## [55] "Plo Koon" "Mas Amedda" "Gregar Typho"   
## [58] "Cordé" "Cliegg Lars" "Poggle the Lesser"   
## [61] "Luminara Unduli" "Barriss Offee" "Dormé"   
## [64] "Dooku" "Bail Prestor Organa" "Jango Fett"   
## [67] "Zam Wesell" "Dexter Jettster" "Lama Su"   
## [70] "Taun We" "Jocasta Nu" "Ratts Tyerell"   
## [73] "R4-P17" "Wat Tambor" "San Hill"   
## [76] "Shaak Ti" "Grievous" "Tarfful"   
## [79] "Raymus Antilles" "Sly Moore" "Tion Medon"   
## [82] "Finn" "Rey" "Poe Dameron"   
## [85] "BB8" "Captain Phasma" "Padmé Amidala"

Try the same with “height” and map\_dbl because you’d expect height to be numeric:

map\_dbl(sw\_people, "height")

## Error in `map\_dbl()`:  
## ℹ In index: 1.  
## Caused by error:  
## ! Can't coerce from a string to a double vector.

If you map “height” as a string you would soon see why is that:

map\_chr(sw\_people, "height")

## [1] "172" "167" "96" "202" "150" "178" "165"   
## [8] "97" "183" "182" "188" "180" "228" "180"   
## [15] "173" "175" "170" "180" "66" "170" "183"   
## [22] "200" "190" "177" "175" "180" "150" "unknown"  
## [29] "88" "160" "193" "191" "170" "196" "224"   
## [36] "206" "183" "137" "112" "183" "163" "175"   
## [43] "180" "178" "94" "122" "163" "188" "198"   
## [50] "196" "171" "184" "188" "264" "188" "196"   
## [57] "185" "157" "183" "183" "170" "166" "165"   
## [64] "193" "191" "183" "168" "198" "229" "213"   
## [71] "167" "79" "96" "193" "191" "178" "216"   
## [78] "234" "188" "178" "206" "unknown" "unknown" "unknown"  
## [85] "unknown" "unknown" "165"

We could take a general approach like this:

chars <- tibble(name = map\_chr(sw\_people, "name"),  
 gender = map\_chr(sw\_people, "gender"),  
 height = map\_chr(sw\_people, "height"))  
chars

## # A tibble: 87 × 3  
## name gender height  
## <chr> <chr> <chr>   
## 1 Luke Skywalker male 172   
## 2 C-3PO n/a 167   
## 3 R2-D2 n/a 96   
## 4 Darth Vader male 202   
## 5 Leia Organa female 150   
## 6 Owen Lars male 178   
## 7 Beru Whitesun lars female 165   
## 8 R5-D4 n/a 97   
## 9 Biggs Darklighter male 183   
## 10 Obi-Wan Kenobi male 182   
## # ℹ 77 more rows

But this would tire us out fast, and can be quite slow with large lists (mapping the same large list over and over again, each time for a single attribute). What I usually do is write a function extracting each “row” out of a list element, then use map\_dfr() to map all list elements with my function to a data.frame:

get\_sw\_person\_details <- function(person) {  
 list(  
 name = person$name,  
 gender = person$gender,  
 height = person$height  
 )  
}  
  
chars <- sw\_people %>% map\_dfr(get\_sw\_person\_details)  
chars

But this didn’t really solve the problem, we’re still manually iterating over attributes.

You should know the pluck() and chuck() functions from purrr for extracting a single element of a list:

l = list(a = 1)  
  
pluck(l, "a")

## [1] 1

chuck(l, "a")

## [1] 1

pluck(l, "b")

## NULL

chuck(l, "b")

## Error in `chuck()`:  
## ! Can't find name `b` in vector.

As you can see chuck() is **safer** as it returns an error in case the element does not exist.

If you want to be a bit more concise (and if you have many attributes) you could use a vector of attributes c("name", "gender", "height") and map a function like pluck() or chuck() to get all attributes. I will use chuck():

char\_attrs <- c("name", "gender", "height", "mass")  
  
get\_sw\_element\_details <- function(element, attrs) {  
 element\_list <- map(attrs, function(attribute) chuck(element, attribute))  
 names(element\_list) <- attrs  
 element\_list  
}  
  
chars <- sw\_people %>% map\_dfr(get\_sw\_element\_details, char\_attrs)  
chars

## # A tibble: 87 × 4  
## name gender height mass   
## <chr> <chr> <chr> <chr>  
## 1 Luke Skywalker male 172 77   
## 2 C-3PO n/a 167 75   
## 3 R2-D2 n/a 96 32   
## 4 Darth Vader male 202 136   
## 5 Leia Organa female 150 49   
## 6 Owen Lars male 178 120   
## 7 Beru Whitesun lars female 165 75   
## 8 R5-D4 n/a 97 32   
## 9 Biggs Darklighter male 183 84   
## 10 Obi-Wan Kenobi male 182 77   
## # ℹ 77 more rows

Do you see why this function is much less prone to future bugs and much more usable in the future, e.g. with sw\_films?

But what will happen once we add in an attribute which not all SW characters have, like “species”?

char\_attrs <- c(char\_attrs, "species")  
  
chars <- sw\_people %>% map\_dfr(get\_sw\_element\_details, char\_attrs)

## Error in `map()`:  
## ℹ In index: 37.  
## Caused by error in `map()`:  
## ℹ In index: 5.  
## Caused by error in `chuck()`:  
## ! Can't find name `species` in vector.

Because “species” and other attributes do not appear for all SW characters chuck() throws an error.

We can make a safer version of chuck() by using one of purrr’s wrapper functions, e.g. possibly(), which lets you specify a value to be returned in case of an error:

chuck\_safely <- possibly(chuck, otherwise = NA)  
  
get\_sw\_element\_details <- function(element, attrs) {  
 element\_list <- map(attrs, function(attr) chuck\_safely(element, attr))  
 names(element\_list) <- attrs  
 element\_list  
}  
  
chars <- sw\_people %>% map\_dfr(get\_sw\_element\_details, char\_attrs)  
chars

## # A tibble: 87 × 5  
## name gender height mass species   
## <chr> <chr> <chr> <chr> <chr>   
## 1 Luke Skywalker male 172 77 http://swapi.co/api/species/1/  
## 2 C-3PO n/a 167 75 http://swapi.co/api/species/2/  
## 3 R2-D2 n/a 96 32 http://swapi.co/api/species/2/  
## 4 Darth Vader male 202 136 http://swapi.co/api/species/1/  
## 5 Leia Organa female 150 49 http://swapi.co/api/species/1/  
## 6 Owen Lars male 178 120 http://swapi.co/api/species/1/  
## 7 Beru Whitesun lars female 165 75 http://swapi.co/api/species/1/  
## 8 R5-D4 n/a 97 32 http://swapi.co/api/species/2/  
## 9 Biggs Darklighter male 183 84 http://swapi.co/api/species/1/  
## 10 Obi-Wan Kenobi male 182 77 http://swapi.co/api/species/1/  
## # ℹ 77 more rows

We’re still not done! Look what happens when we add “vehicles”:

char\_attrs <- c(char\_attrs, "vehicles")  
  
chars <- sw\_people %>% map\_dfr(get\_sw\_element\_details, char\_attrs)

In previous versions of tidyverse this resulted in an error, because the “vehicles” field is in itself a list of a few vehicles “ID”s, which threw map\_dfr() off the rails, where we *would* have wanted a list column (column holding lists of varying length of vehicles). In the most recent version it silently un-nests the vehicles from each list into multiple rows, which I think is worse, but it is debatable.

Here we have a few options. I prefer keeping a **named** vector of chars\_attrs (like a Python dictionary), specifying for each attribute whether a list is expected or not, then choosing what to do with such attribute in another function, chuck\_safest():

char\_attrs <- c("name" = FALSE, "gender" = FALSE, "height" = FALSE, "mass" = FALSE, "species" = FALSE, "vehicles" = TRUE)  
  
chuck\_safest <- function(element, attr, attrs\_dict) {  
 res <- chuck\_safely(element, attr)  
 if (attrs\_dict[attr]) {  
 return(list(res))  
 }  
 return(res)  
}  
get\_sw\_element\_details <- function(element, attrs\_dict) {  
 attrs <- names(attrs\_dict)  
 element\_list <- map(attrs, function(attr) chuck\_safest(element, attr, attrs\_dict))  
 names(element\_list) <- attrs  
 element\_list  
}  
  
chars <- sw\_people %>% map\_dfr(get\_sw\_element\_details, char\_attrs)  
chars

## # A tibble: 87 × 6  
## name gender height mass species vehicles  
## <chr> <chr> <chr> <chr> <chr> <list>   
## 1 Luke Skywalker male 172 77 http://swapi.co/api/species/… <chr>   
## 2 C-3PO n/a 167 75 http://swapi.co/api/species/… <lgl>   
## 3 R2-D2 n/a 96 32 http://swapi.co/api/species/… <lgl>   
## 4 Darth Vader male 202 136 http://swapi.co/api/species/… <lgl>   
## 5 Leia Organa female 150 49 http://swapi.co/api/species/… <chr>   
## 6 Owen Lars male 178 120 http://swapi.co/api/species/… <lgl>   
## 7 Beru Whitesun lars female 165 75 http://swapi.co/api/species/… <lgl>   
## 8 R5-D4 n/a 97 32 http://swapi.co/api/species/… <lgl>   
## 9 Biggs Darklighter male 183 84 http://swapi.co/api/species/… <lgl>   
## 10 Obi-Wan Kenobi male 182 77 http://swapi.co/api/species/… <chr>   
## # ℹ 77 more rows

Look at the vehicles column. It is a list column, as we wanted, but we can see that some lists are marked “<chr[2]>” and some “<lgl[1]>”, which means some are of type “character” and some “logical”. This happens because the NA value returned by chuck\_safely has by default type “logical”!

typeof(NA)

## [1] "logical"

And so one final touch would be to make chuck\_safely() return a “character” NA explicitly, like so:

chuck\_safely <- possibly(chuck, otherwise = NA\_character\_)

And now:

chars <- sw\_people %>% map\_dfr(get\_sw\_element\_details, char\_attrs)  
chars

## # A tibble: 87 × 6  
## name gender height mass species vehicles  
## <chr> <chr> <chr> <chr> <chr> <list>   
## 1 Luke Skywalker male 172 77 http://swapi.co/api/species/… <chr>   
## 2 C-3PO n/a 167 75 http://swapi.co/api/species/… <chr>   
## 3 R2-D2 n/a 96 32 http://swapi.co/api/species/… <chr>   
## 4 Darth Vader male 202 136 http://swapi.co/api/species/… <chr>   
## 5 Leia Organa female 150 49 http://swapi.co/api/species/… <chr>   
## 6 Owen Lars male 178 120 http://swapi.co/api/species/… <chr>   
## 7 Beru Whitesun lars female 165 75 http://swapi.co/api/species/… <chr>   
## 8 R5-D4 n/a 97 32 http://swapi.co/api/species/… <chr>   
## 9 Biggs Darklighter male 183 84 http://swapi.co/api/species/… <chr>   
## 10 Obi-Wan Kenobi male 182 77 http://swapi.co/api/species/… <chr>   
## # ℹ 77 more rows

### Iterating with purrr - what you need to do

##### (50 points)

First, from the sw\_films list, create a films data.frame, holding for each SW film a single row with variables c("title", "episode\_id", "director", "release\_date", "characters", "planets", "starships" and "vehicles"). Notice:

* some of these variables have a single element, some are vectors of elements
* there’s *very* little you need to do if you followed along and you don’t rewrite functions already written…

char\_attrs <- c("title"= FALSE, "episode\_id"= FALSE, "director"= FALSE, "release\_date"= FALSE, "characters"= TRUE, "planets"= TRUE, "starships"= TRUE, "vehicles"= TRUE)  
  
films <- sw\_films %>% map\_dfr(get\_sw\_element\_details, char\_attrs)  
films

## # A tibble: 7 × 8  
## title episode\_id director release\_date characters planets starships vehicles  
## <chr> <int> <chr> <chr> <list> <list> <list> <list>   
## 1 A New … 4 George … 1977-05-25 <chr [18]> <chr> <chr [8]> <chr>   
## 2 Attack… 2 George … 2002-05-16 <chr [40]> <chr> <chr [9]> <chr>   
## 3 The Ph… 1 George … 1999-05-19 <chr [34]> <chr> <chr [5]> <chr>   
## 4 Reveng… 3 George … 2005-05-19 <chr [34]> <chr> <chr> <chr>   
## 5 Return… 6 Richard… 1983-05-25 <chr [20]> <chr> <chr> <chr>   
## 6 The Em… 5 Irvin K… 1980-05-17 <chr [16]> <chr> <chr [9]> <chr>   
## 7 The Fo… 7 J. J. A… 2015-12-11 <chr [11]> <chr> <chr [2]> <chr>

Now, the chars dataset isn’t perfect:

* Redo it, adding variables “birth\_year”, “homeworld” and “films”
* Missing values still appear as “n/a” or “unknown” - they need to be NA
* Columns height and mass are still of type character - they need to be numeric
* Change column “homeworld” to have its integer ID rather than the entire planet URL (e.g. instead of “<http://swapi.co/api/planets/1/>” there should be 1)
* Bonus: can you do this for each of the “films” list?
* The “birth\_year” column holds string values such as “19BBY”. It should be just “19” and numeric.
* Add a column character\_id numbering each character from 1 to last, and make it the *first* column of the dataset

You may write functions to help you but I would prefer if the actual cleaning would be performed in a single pipe.

You must use at least one purrr mapping function such as map\_dbl().

char\_attrs <- c("name" = FALSE, "gender" = FALSE, "height" = FALSE, "mass" = FALSE, "species" = FALSE, "vehicles" = TRUE, "birth\_year" = FALSE, "homeworld" = FALSE, films = TRUE)  
chars <- sw\_people %>% map\_dfr(get\_sw\_element\_details, char\_attrs)  
  
chars <- chars %>%   
 mutate(  
 across(c(height, mass), ~if\_else(. %in% c("n/a", "unknown"), NA\_real\_, as.numeric(.))),  
 gender = if\_else(gender %in% c("n/a", "unknown"), NA\_character\_, gender),  
 homeworld = gsub("[^0-9]+", "", homeworld),  
 birth\_year = gsub("[^0-9]+", "", birth\_year),  
 character\_id = row\_number()  
 ) %>%   
 select(character\_id, everything())

## Warning: There were 2 warnings in `mutate()`.  
## The first warning was:  
## ℹ In argument: `across(...)`.  
## Caused by warning in `if\_else()`:  
## ! NAs introduced by coercion  
## ℹ Run ]8;;ide:run:dplyr::last\_dplyr\_warnings()dplyr::last\_dplyr\_warnings()]8;; to see the 1 remaining warning.

chars

## # A tibble: 87 × 10  
## character\_id name gender height mass species vehicles birth\_year homeworld  
## <int> <chr> <chr> <dbl> <dbl> <chr> <list> <chr> <chr>   
## 1 1 Luke … male 172 77 http:/… <chr> "19" 1   
## 2 2 C-3PO <NA> 167 75 http:/… <chr> "112" 1   
## 3 3 R2-D2 <NA> 96 32 http:/… <chr> "33" 8   
## 4 4 Darth… male 202 136 http:/… <chr> "419" 1   
## 5 5 Leia … female 150 49 http:/… <chr> "19" 2   
## 6 6 Owen … male 178 120 http:/… <chr> "52" 1   
## 7 7 Beru … female 165 75 http:/… <chr> "47" 1   
## 8 8 R5-D4 <NA> 97 32 http:/… <chr> "" 1   
## 9 9 Biggs… male 183 84 http:/… <chr> "24" 1   
## 10 10 Obi-W… male 182 77 http:/… <chr> "57" 20   
## # ℹ 77 more rows  
## # ℹ 1 more variable: films <list>

# bonus: change films  
chars %>% mutate(films = (gsub("[^0-9]+", "", films)))

## # A tibble: 87 × 10  
## character\_id name gender height mass species vehicles birth\_year homeworld  
## <int> <chr> <chr> <dbl> <dbl> <chr> <list> <chr> <chr>   
## 1 1 Luke … male 172 77 http:/… <chr> "19" 1   
## 2 2 C-3PO <NA> 167 75 http:/… <chr> "112" 1   
## 3 3 R2-D2 <NA> 96 32 http:/… <chr> "33" 8   
## 4 4 Darth… male 202 136 http:/… <chr> "419" 1   
## 5 5 Leia … female 150 49 http:/… <chr> "19" 2   
## 6 6 Owen … male 178 120 http:/… <chr> "52" 1   
## 7 7 Beru … female 165 75 http:/… <chr> "47" 1   
## 8 8 R5-D4 <NA> 97 32 http:/… <chr> "" 1   
## 9 9 Biggs… male 183 84 http:/… <chr> "24" 1   
## 10 10 Obi-W… male 182 77 http:/… <chr> "57" 20   
## # ℹ 77 more rows  
## # ℹ 1 more variable: films <chr>

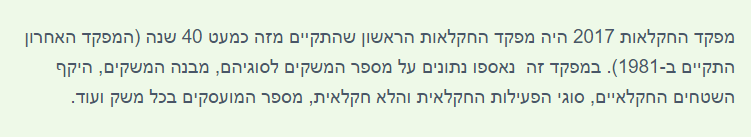
Finally, I just want you to acknowledge the fact that we didn’t use a single for loop. For loops are not necessarily “bad” or “slow” but there are available solutions like purrr, that allow you to write more concise, less error-prone code. Make yourself love it, it will pay off.

### Paper questions

##### (10 points)

Read Broman and Woo [Data Organization in Spreadsheets](https://www.tandfonline.com/doi/full/10.1080/00031305.2017.1375989) 2017 paper. It is actually one of The American Statistician journal most viewed papers.

Look at the משקים חקלאיים attached Excel file in the data folder. This is an actual file from the Israeli Central Bureau of Statistics, and reflects quite well how files originating from the CBS usually look like.



Which Broman and Woo principles are violated in this file? Give at least 4 and explain each. You can add a few from your own experience if you like.

# principle 1: Choose Good Names for Things  
# explanation - the names of the observations contain spaces, for example "jewish - other"  
  
# principle 2: no empty cells   
# explanation - D11 is an empty cell  
  
# principle 3: create a data dictionary  
# explanation - the data does not have a separate file that explains what all of the variables are.  
  
# principle 4: Do Not Use Font Color or Highlighting as Data  
# explanation - all of the districts rows are highlighted

### Wrap up

And that’s it, you have shown you can clean a messy Excel sheet and tidy up a complex JSON API list into a nice table you can actually analyze. Good luck with the rest of the course!