JSON

02-18-2020

Getting started with JSON

```
library(tidyverse)
```

```
## -- Attaching packages ------- tidyverse 1.3.0 --

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ---------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(jsonlite)
```

```
##
## Attaching package: 'jsonlite'

## The following object is masked from 'package:purrr':
##
##     flatten
```

JavaScript Object Notation (JSON) is an open-standard file format or data interchange format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
```

```
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

In JSON, values must be one of the following data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- null

Parsing JSON

```
txt <- "[12, 3, 7]"
x <- fromJSON(txt)
x
```

```
## [1] 12  3  7
```

```
# from file
fromJSON("somefile.json")
# from the web
fromJSON("https://www.example.com/hello.json")
```

Simplification

Simplification is the process where JSON arrays automatically get converted from a list into a more specific R class. The `fromJSON` function has 3 arguments which control the simplification process: `simplifyVector`, `simplifyDataFrame` and `simplifyMatrix`. Each one is enabled by default.

JSON structure	Example JSON data	Simplifies to R class	Argument in fromJSON
Array of primitives	`["Amsterdam", "Rotterdam", "Utrecht", "Den Haag"]`	Atomic Vector	simplifyVector
Array of objects	`[{"name":"Erik", "age":43}, {"name":"Anna", "age":32}]`	Data Frame	simplifyDataFrame
Array of arrays	`[ [1, 2, 3], [4, 5, 6] ]`	Matrix	simplifyMatrix

Atomic Vectors

When `simplifyVector` is enabled, JSON arrays containing **primitives** (strings, numbers, booleans or null) simplify into an atomic vector:

```
# A JSON array of primitives
json <- '["Mario", "Peach", null, "Bowser"]'
# Simplifies into an atomic vector
fromJSON(json)
```

```
## [1] "Mario"  "Peach"  NA       "Bowser"
```

Without simplification, any JSON array turns into a list:

```
# No simplification:
fromJSON(json, simplifyVector = FALSE)
```

```
## [[1]]
## [1] "Mario"
##
## [[2]]
## [1] "Peach"
##
## [[3]]
## NULL
##
## [[4]]
## [1] "Bowser"
```

Data Frames

When `simplifyDataFrame` is enabled, JSON arrays containing **objects** (key-value pairs) simplify into a data frame:

```
json <-
'[
  {"Name" : "Mario", "Age" : 32, "Occupation" : "Plumber"},
  {"Name" : "Peach", "Age" : 21, "Occupation" : "Princess"},
  {},
  {"Name" : "Bowser", "Occupation" : "Koopa"}
]'
mydf <- fromJSON(json)
mydf
```

```
##     Name Age Occupation
## 1  Mario  32    Plumber
## 2  Peach  21   Princess
## 3   <NA>  NA       <NA>
## 4 Bowser  NA      Koopa
```

The data frame gets converted back into the original JSON structure by `toJSON` (whitespace and line breaks are ignorable in JSON).

```
mydf$Ranking <- c(3, 1, 2, 4)
toJSON(mydf, pretty=TRUE)
```

```
## [
##   {
##     "Name": "Mario",
##     "Age": 32,
##     "Occupation": "Plumber",
##     "Ranking": 3
##   },
##   {
##     "Name": "Peach",
##     "Age": 21,
##     "Occupation": "Princess",
##     "Ranking": 1
##   },
##   {
##     "Ranking": 2
##   },
##   {
##     "Name": "Bowser",
##     "Occupation": "Koopa",
##     "Ranking": 4
##   }
## ]
```

Hence you can go back and forth between dataframes and JSON, without any manual data restructuring.

Matrices and Arrays

When `simplifyMatrix` is enabled, JSON arrays containing **equal-length sub-arrays** simplify into a matrix (or higher order R array):

```
json <- '[
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9, 10, 11, 12]
]'
mymatrix <- fromJSON(json)
mymatrix
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

Again, we can use `toJSON` to convert the matrix or array back into the original JSON structure:

```
toJSON(mymatrix, pretty = TRUE)
```

```
## [
##   [1, 2, 3, 4],
##   [5, 6, 7, 8],
##   [9, 10, 11, 12]
## ]
```

The simplification works for arrays of arbitrary dimensionality, as long as the dimensions match (R does not support ragged arrays).

```
json <- '[
   [[1, 2],
    [3, 4]],
   [[5, 6],
    [7, 8]],
   [[9, 10],
    [11, 12]]
]'
myarray <- fromJSON(json)
myarray[1, , ]
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

```
myarray[ , ,1]
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    5    7
## [3,]    9   11
```

API

This section lists some examples of public HTTP APIs that publish data in JSON format. These are great to get a sense of the complex structures that are encountered in real world JSON data.

CitiBike NYC

A single public API that shows location, status and current availability for all stations in the New York City bike sharing imitative. https://www.citibikenyc.com/system-data

```
citibike <- fromJSON("https://gbfs.citibikenyc.com/gbfs/en/station_status.json")
stations <- citibike$data$stations
colnames(stations)
```

```
##  [1] "station_id"                    "num_bikes_available"
##  [3] "num_ebikes_available"          "num_bikes_disabled"
##  [5] "num_docks_available"           "num_docks_disabled"
##  [7] "is_installed"                  "is_renting"
##  [9] "is_returning"                  "last_reported"
## [11] "eightd_has_available_keys"     "eightd_active_station_services"
```

```
nrow(stations)
```

```
## [1] 935
```

ProPublica

Below an example from the ProPublica Nonprofit Explorer API where we retrieve the first 10 pages of tax-exempt organizations in the USA, ordered by revenue. The `rbind_pages` function is used to combine the pages into a single data frame.

```
#store all pages in a list first
baseurl <- "https://projects.propublica.org/nonprofits/api/v2/search.json?order=revenue&sort_order=desc
pages <- list()
for(i in 0:9){
  mydata <- fromJSON(paste0(baseurl, "&page=", i), flatten=TRUE)
  message("Retrieving page ", i)
  pages[[i+1]] <- mydata$organizations
}
```

```
## Retrieving page 0

## Retrieving page 1

## Retrieving page 2

## Retrieving page 3

## Retrieving page 4

## Retrieving page 5

## Retrieving page 6

## Retrieving page 7

## Retrieving page 8

## Retrieving page 9
```

```
#combine all into one
organizations <- rbind_pages(pages)
#check output
nrow(organizations)
```

```
## [1] 1000
```

```
organizations %>% head(10) %>% select(name, city, state)
```

```
##                               name        city state
## 1           0 DEBT EDUCATION INC  SANTA ROSA    CA
## 2                0 TOLERANCE INC     SUWANEE    GA
## 3                  0 U R PASSION   KENNEWICK    WA
## 4               00 MOVEMENT INC   PENSACOLA    FL
## 5                   00006 LOCAL       MEDIA    PA
## 6             0003 POSTAL FAMILY  CINCINNATI    OH
## 7                        0005 GA   HEPHZIBAH    GA
## 8  0005 WRIGHT-PATT CREDIT UNION BEAVERCREEK    OH
## 9                        0009 DE   GREENWOOD    DE
## 10              0011 CALIFORNIA      REDWAY    CA
```

Reference

jsonlite quick start: https://cran.r-project.org/web/packages/jsonlite/vignettes/json-aaquickstart.html jsonlite apis: https://cran.r-project.org/web/packages/jsonlite/vignettes/json-apis.html