File Representations of Symbolic Data for Open Science

Vladimir Batagelj

IMFM Ljubljana, IAM & FAMNIT UP Koper, FMF UL Ljubljana

Symbolic Data Analysis (SDA) Workshop 2025

June 9–11, 2025, Varaždin, Croatia

Outline

Vladimir Batagelj: vladimir.batagelj@fmf.uni-lj.si
Current version of slides (June 9, 2025 at 01 : 58): slides PDF
https://github.com/bavla/symData/

Open science

In Open Science (Wikipedia, 2025), there is a growing emphasis on publishing research data following the FAIR principles (Findable, Accessible, Interoperable, Reusable) (GoFAIR, 2016). Adhering to these standards ensures the verifiability of results and enables alternative analyses. Additionally, open data contributes to greater diversity in datasets, supporting the development and testing of new methodologies.

In symbolic data analysis, the starting point is usually a generalized (symbolic) data table, where variable values can be structured (combinations of primitive values). These require specialized external (file-based) and internal (in-memory) representations. Ideally, the two representations would be compatible.

This presentation focuses on file-based descriptions of symbolic data tables, which can facilitate seamless data exchange between symbolic data analysis tools (based on different prog. languages).

Data tables

Traditional data analysis is based on a (simple) *data table* $\mathbf{T}_{U \times V}$, over a set of *units* U and a set of unit properties or *variables* V. The entry $T[u, v]$ contains the (measured) *value* of a property $v \in V$ at a unit $u \in U$. The values are simple data: numbers, logical values, dates, and character strings.

LOREOM IPSUMAT CONSTRUM ETRIM KIST	Lorem ipsum dolor	Amister umarkl finish	Gatolep odio un accums	Tortores remus justica
LOREM DOLOR SIAMET	8 288	123 %	YES	$89
CONSECTER ODIO	123	87 %	NO	$129
GATOQUE ACCUMS	1 005	12 %	NO	$99
SED HAC ENIM REM	56	69 %	N/A	$199
REMPUS TORTOR JUST	5 554	18 %	NO	$999
FCELISQUE SED MORBI	12 569	112 %	NO	$123
SENECTUS URNA MOSTUM	779	33 %	N/A	$56
VESIBU LORIS SET MURTIL	6 112	27 %	YES	$684

Data tables

When encoding data, there may be a need for unusual values, such as unknown, meaningless, or infinite.

Spreadsheet programs, such as Excel, can be used to prepare, maintain, and perform simple analyses of such tables.

In recent times, there are more and more examples of data that go beyond simple tables - the values can be composite data: time series, sequences of events, sets of strings, intervals, distributions, graphs, etc.

Sometimes we add one or more (weighted) relations between the units – we get a network. If we convert the table **T** into triples $(u, v, T[u, v])$, we get a knowledge graph.

Generalized tables in R

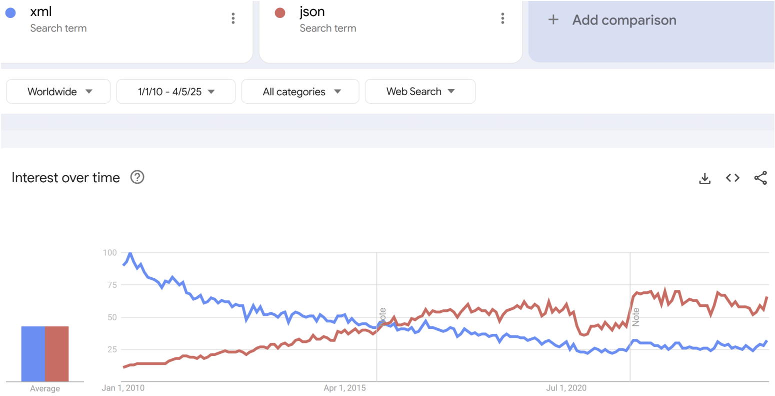
In classical data analysis, for saving/reading the data, the CSV (Comma Separated Values) format is usually used. It is not appropriate for structured data because it doesn't preserve the structure.

While working in the R environment, for saving and data exchange, we can use the RDS files ("R Data Serialization" or also "R Dataset Single"). They preserve the content of the data object (`saveRDS` and `readRDS`). For larger data objects, the saved data can be compressed.

For exchange between different software environments (R, Python, Julia, C++, Javascript, etc.) and archiving, we have to use a structure-and-data preserving text format. The standard options are XML and JSON.

The World Factbook: WWW, GitHub/JSON, Kaggle, Ian Coleman.

Google trends XML : JSON



Google trends XML API : JSON API

Google trends XML : JSON

Most formats for structured data are based on XML or JSON, with JSON increasingly favored in modern applications – see Figure.

JSON description is not only a valid JavaScript expression but also uses data structures that are natively supported by most programming languages (e.g., *R*, *Python*, *Julia*, *C++*) (JSON, 2017; ECMAScript, 2024; Batagelj, 2016).

To understand why JSON is lightweight, consider the representation of a person in XML and it's JSON equivalent:

XML:

```
<person>
<first-name>Janez</first-name>
<last-name>Novak</last-name>
</person>
```

JSON:

```
{
"firstname":"Janez",
"lastname":"Novak"
}
```

JSON

The basic idea of the JSON (JavaScript Object Notation) format (RFC 8259 and ECMA-404) is that in JavaScript, a JSON data description is evaluated into a JavaScript data value (object).

The JSON description of a data object starts with atomic data values (numbers, strings, logical values, dates) and combines already constructed data objects using either an ordered list

[dob1, dob2, ..., dobk]

or a named (unordered) list

{ ”nam1”: dob1, ”nam2”: dob2, ..., ”namk”: dobk }

```
[{"name":"Anna","sex":"F","age":29,
  "phone":{"home":"051123456","work":"051123456"}},
 {"name":"Charles","sex":"M","age":35,
   "phone":{"work":["051987654","051456789"]}} ]
```

SDA JSON – first attempt

Five years ago I created Billard-Diday's symbolic data sets in JSON. For example: Joggers data

```
{ "SDA": "manual",
  "info": {
    "source": "Lynne Billard, Edwin Diday: Clustering Methodology for Symbolic Data,
    "title": "Joggers / Mixed-valued data"
  },
  "vars": [ ["id", "Group"], ["Y1","Pulse rate"], ["Y2","Running time"]],
  "units": [
    [ 1, [73, 114] , [ [[5.3, 6.2], 0.3], [[6.2, 7.1], 0.5], [[7.1, 8.3], 0.2]] ],
    [ 2, [70, 100] , [ [[5.5, 6.9], 0.4], [[6.7, 8.0], 0.4], [[8.0, 9.0], 0.2]] ],
    [ 3, [69, 91] , [ [[5.1, 6.6], 0.4], [[6.6, 7.4], 0.4], [[7.4, 7.8], 0.2]] ],
    [ 4, [59, 89] , [ [[3.7, 5.8], 0.6], [[5.8, 6.3], 0.4]] ],
    [ 5, [61, 87] , [ [[4.5, 5.9], 0.4], [[5.9, 6.2], 0.6]] ],
    [ 6, [69, 95] , [ [[4.1, 6.1], 0.5], [[6.1, 6.9], 0.5]] ],
    [ 7, [65, 78] , [ [[2.4, 4.8], 0.3], [[4.8, 5.7], 0.5], [[5.7, 6.2], 0.2]] ],
    [ 8, [58, 83] , [ [[2.1, 5.4], 0.2], [[5.4, 6.0], 0.5], [[6.0, 6.9], 0.3]] ],
    [ 9, [79, 103] , [ [[4.8, 6.5], 0.3], [[6.5, 7.4], 0.5], [[7.4, 8.2], 0.2]] ],
    [10, [40, 60] , [ [[3.2, 4.1], 0.6], [[4.1, 6.7], 0.4]] ]
  ]
}
```

It is relatively easy to convert the corresponding R data object into a symbolic data table. Among available options (`data.frame`, `tibble`, `data.table`), I prefer data.table.

SDA JSON – trying to avoid conversion


```
{ "SDA": "manual",
  "info": {
    "source": "Lynne Billard, Edwin Diday: Clustering Methodology for Symbolic Data,
    "title": "Joggers / extended example",
    "nUnits": 10, "nVars": 5,
    "trace": [{"by": "Vladimir Batagelj", "date": "Sat Jun  7 03:01:17 2025"}]
  },
  "vars": [ ["name","Person","chr"], ["w","Weight","num"], ["comp","Type of","seq"],
      ["S","Pulse rate","ivl"], ["H","Running time","his"]],
  "units": [
    {"name":"A","w":81,"comp":["Y","Y","Z","X","X"],"S":[73,114],
     "H.lb":[5.3,6.2,7.1],"H.rb":[6.2,7.1,8.3],"H.p":[0.3,0.5,0.2]},
    {"name":"B","w":88,"comp":["Y","Y","Z","X"],"S":[70,100],
     "H.lb":[5.5,6.7,8],"H.rb":[6.9,8,9],"H.p":[0.4,0.4,0.2]},
    {"name":"C","w":61,"comp":["Z","Z"],"S":[69,91],
     "H.lb":[5.1,6.6,7.4],"H.rb":[6.6,7.4,7.8],"H.p":[0.4,0.4,0.2]},
    {"name":"D","w":68,"comp":["X","Z","X","Z"],"S":[59,89],
     "H.lb":[3.7,5.8],"H.rb":[5.8,6.3],"H.p":[0.6,0.4]},
    {"name":"E","w":83,"comp":["Z","X","X"],"S":[61,87],
     "H.lb":[4.5,5.9],"H.rb":[5.9,6.2],"H.p":[0.4,0.6]},
    {"name":"F","w":71,"comp":["X"],"S":[69,95],
     "H.lb":[4.1,6.1],"H.rb":[6.1,6.9],"H.p":[0.5,0.5]},
    {"name":"G","w":70,"comp":["Y","X"],"S":[65,78],
     "H.lb":[2.4,4.8,5.7],"H.rb":[4.8,5.7,6.2],"H.p":[0.3,0.5,0.2]},
    {"name":"H","w":62,"comp":["X"],"S":[58,83],
     "H.lb":[2.1,5.4,6],"H.rb":[5.4,6,6.9],"H.p":[0.2,0.5,0.3]},
    {"name":"I","w":80,"comp":["Z","Y","Z"],"S":[79,103],
     "H.lb":[4.8,6.5,7.4],"H.rb":[6.5,7.4,8.2],"H.p":[0.3,0.5,0.2]},
    {"name":"J","w":80,"comp":["Y","Z"],"S":[40,60],
     "H.lb":[3.2,4.1],"H.rb":[4.1,6.7],"H.p":[0.6,0.4]}
  ]
}
```

SDA JSON – it works


```
> Jex <- fromJSON("JoggersExt.json")
> names(Jex)
[1] "SDA"    "info"   "vars"   "units"
> Jex$info$title
[1] "Joggers / extended example"
> Jex$info$trace
                 by                       date
1 Vladimir Batagelj Sat Jun  7 03:01:17 2025
> Jex$vars
     [,1]   [,2]           [,3]
[1,] "name" "Person"       "chr"
[2,] "w"    "Weight"       "num"
[3,] "comp" "Type of"      "seq"
[4,] "S"    "Pulse rate"   "ivl"
[5,] "H"    "Running time" "his"
> D <- as.data.table(Jex$units)
> D
      name     w      comp      S        H.lb        H.rb         H.p
    <char> <int>    <list> <list>      <list>      <list>      <list>
 1:      A    81 Y,Y,Z,X,X 73,114 5.3,6.2,7.1 6.2,7.1,8.3 0.3,0.5,0.2
 2:      B    88   Y,Y,Z,X 70,100 5.5,6.7,8.0 6.9,8.0,9.0 0.4,0.4,0.2
 3:      C    61       Z,Z  69,91 5.1,6.6,7.4 6.6,7.4,7.8 0.4,0.4,0.2
 4:      D    68   X,Z,X,Z  59,89     3.7,5.8     5.8,6.3     0.6,0.4
 5:      E    83     Z,X,X  61,87     4.5,5.9     5.9,6.2     0.4,0.6
 6:      F    71         X  69,95     4.1,6.1     6.1,6.9     0.5,0.5
 7:      G    70       Y,X  65,78 2.4,4.8,5.7 4.8,5.7,6.2 0.3,0.5,0.2
 8:      H    62         X  58,83 2.1,5.4,6.0 5.4,6.0,6.9 0.2,0.5,0.3
 9:      I    80     Z,Y,Z 79,103 4.8,6.5,7.4 6.5,7.4,8.2 0.3,0.5,0.2
10:      J    80       Y,Z  40,60     3.2,4.1     4.1,6.7     0.6,0.4
> D[3,H.p][[1]]
[1] 0.4 0.4 0.2
```

JSON and special values

There are two problems related to numerical values

- most programming languages support the IEEE 754 IEEE Standard for Floating-Point Arithmetic that includes (section 6) also special values **Infinity**, **-Infinity**, and **Not_a_Number** (`+Inf`, `-Inf`, `NaN`). JavaScript allows numbers of unlimited precision, but doesn't support these special values.
- in data analysis, the value **Not_Available** (`NA`) is used to indicate a missing value

See also: Infinity and JSON; JSON status in ECMAScript; JSON in Python 3; Issue 98.

...JSON and special values

The new Javascript standard ECMAScript® 2026 finally introduced (section 6.1.6.1.) values **+Infinity**, **-Infinity**, **NaN**, and **Undefined**.

In R the library `jsonlite` already supports `+Inf`, `-Inf`, `NaN`, and `NA`.

JSON variants:

JSON WP, JSON-LD WP, UBJSON WP, Smile WP

JSON

Inf, NA, NaN

```
> (M <- matrix(c(1:4,NaN,NA,Inf,+Inf,-Inf),ncol=3,byrow=TRUE))
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4  NaN   NA
[3,]  Inf  Inf -Inf
> (m <- toJSON(M))
[[1,2,3],[4,"NaN","NA"],["Inf","Inf","-Inf"]]
> fromJSON(m)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4  NaN   NA
[3,]  Inf  Inf -Inf
> t <- '[[1,2,3],[4,"NaN","NA"],["Inf","Infinity","-Inf"]]'
> fromJSON(t)
     [,1]  [,2]       [,3]
[1,] "1"   "2"        "3"
[2,] "4"   "NaN"      NA
[3,] "Inf" "Infinity" "-Inf"
>
```

JSON tools

Oxygen,

json-editor online, jsonformatter jsoneditor, json-editor, Altova json-tools, jsonlint, json-buddy,

phcode

kate editor Windows

jsoncrack.

Data table format

Beyond the raw data, it is essential to incorporate metadata in the file description. When designing such descriptions, it is advisable to rely on established standards, such as persistent identifiers (DOIs, ORCID, ROR) (DPC, 2025), ISO standards(ISO, 2025), schema.org (Schema, 2025), Dublin Core (DCMI, 2025), etc.

Adopting these practices ensures better interoperability, reusability, and long-term preservation of symbolic data.

Conclusions

1. From a data archiving perspective, any description that preserves the structure of the data is acceptable.
2. The JSON format is easily extensible, allowing for the inclusion of new components.
3. In principle, we have significant freedom in choosing the JSON format for symbolic data. However, if we want to avoid additional conversions, we may need to compromise on some of that flexibility.
4. For sdaJSON to function effectively as an exchange format, its standardization is required among developers of SDA programs.
5. This would involve agreeing on a minimal set of types of symbolic objects and metadata, as well as a consistent naming convention.

Acknowledgments

The computational work reported in this presentation was performed in R. The code and data are available at GitHub/bavla symDATA/format.

This work is partly supported by the Slovenian Research Agency ARIS (research program P1-0294 and research project J5-4596), and prepared within the framework of the COST action CA21163 (HiTEc).

References I

Batagelj, V. (2016). Network visualization based on JSON and D3.js (slides). *Second European Conference on Social Networks. June 14-17, 2016, Paris*. https://github.com/bavla/netsJSON/blob/master/doc/netVis.pdf.

DCMI (3 April 2025). The Dublin Core Metadata Initiative. https://www.dublincore.org/.

Digital Preservation Coalition (2025). Persistent identifiers. In Digital Preservation Handbook. https://www.dpconline.org/handbook/technical-solutions-and-tools/persistent-identifiers.

ECMA (December 2017). Introducing JSON – ECMA-404: The JSON data interchange syntax. 2nd edition. https://www.json.org/json-en.html.

ECMA (June 2024). ECMA-262: ECMAScript® 2024 Language Specification. 15th Edition. https://ecma-international.org/publications-and-standards/standards/ecma-262/.

Go FAIR (2016). FAIR Principles. https://www.go-fair.org/fair-principles/.

ISO (2025). The International Organization for Standardization. https://www.iso.org/.

References II

Schema.org (24 March 2025). Schemas for structured data. https://schema.org/.

Wikipedia (1 April 2025). Open science. https://en.wikipedia.org/wiki/Open_science.