The Journal of Open Source Software

SetVis: Visualizing Large Numbers of Sets and Intersections

R. A. Ruddle [1,2], **L. Hama** [1], **P Wochner** [2], **and O. T. Strickson** [2]

1 University of Leeds, Leeds, United Kingdom **2** Alan Turing Institute, London, United Kingdom

DOI: 10.21105/joss.06925

Software

- Review
- Repository
- Archive

Editor: Marcel Stimberg

Reviewers:

- @kirangadhave
- @jibalamy

Submitted: 10 June 2024
Published: 30 October 2024

Summary

Set-type data occurs in many domains such as life sciences (Lamy & Tsopra, 2019), health (Landolfi et al., 2022) and the retail industry (Adnan & Ruddle, 2018), as well as in generic applications such as analysing structures of missing data (Ruddle, Adnan, & Hall, 2022) and association rule mining (Wang et al., 2020). SetVis is new matrix-based set visualization software, implemented as a Python package which is available from PyPi. The documentation is available from setvis.readthedocs.io which contains various hands-on Jupyter notebooks guiding users to use the package. SetVis uses a memory-efficient design, operates with datasets held in RAM or a database such as PostgreSQL, and allows user interaction graphically or programmatically. A technical evaluation shows that SetVis uses orders of magnitude less memory than the UpSet (Lex et al., 2014) Python package (Nothman, 2022).

Statement of need

Although a wide variety of set visualization software has been developed (Alsallakh & Ren, 2016; Jia et al., 2021), most of such software generates Venn or Euler diagrams so is only suitable for data that contain fewer than ten sets (Jia et al., 2021). Other software visualizes 50+ sets but either has little support for set intersection tasks (Alper et al., 2011; Dörk et al., 2012; Freiler et al., 2008; Kim et al., 2007) or only visualizes pairwise intersections (Molbiotools, 2022; Yalcin et al., 2015).

The best-known software for analysing rich patterns in set data are the R and Python UpSet plot packages (Conway et al., 2017; Nothman, 2022), but the memory requirement of both packages increases linearly with the number of cells (i.e., rows × columns) in a dataset, which makes the packages unusable with big data. The ACE software (Ruddle, Adnan, Kavanagh, et al., 2022) uses more memory-efficient data structures, but first requires the whole of a dataset to have been loaded into RAM (again, clearly an issue for big data), and is a stand-alone Java application that cannot be integrated with Jupyter Notebooks or similar workflows. The SetVis python package addresses the above collective weaknesses because it: (a) operates with datasets that may be either held in RAM or out of core (in a PostgreSQL database), (b) stores sets and intersections in memory-efficient data structures (like ACE), (c) can be used within Jupyter Notebooks (or similar) to aid the replicability of analysis workflows, and (d) allows users to interact graphically in a notebook as well as programmatically.

Design

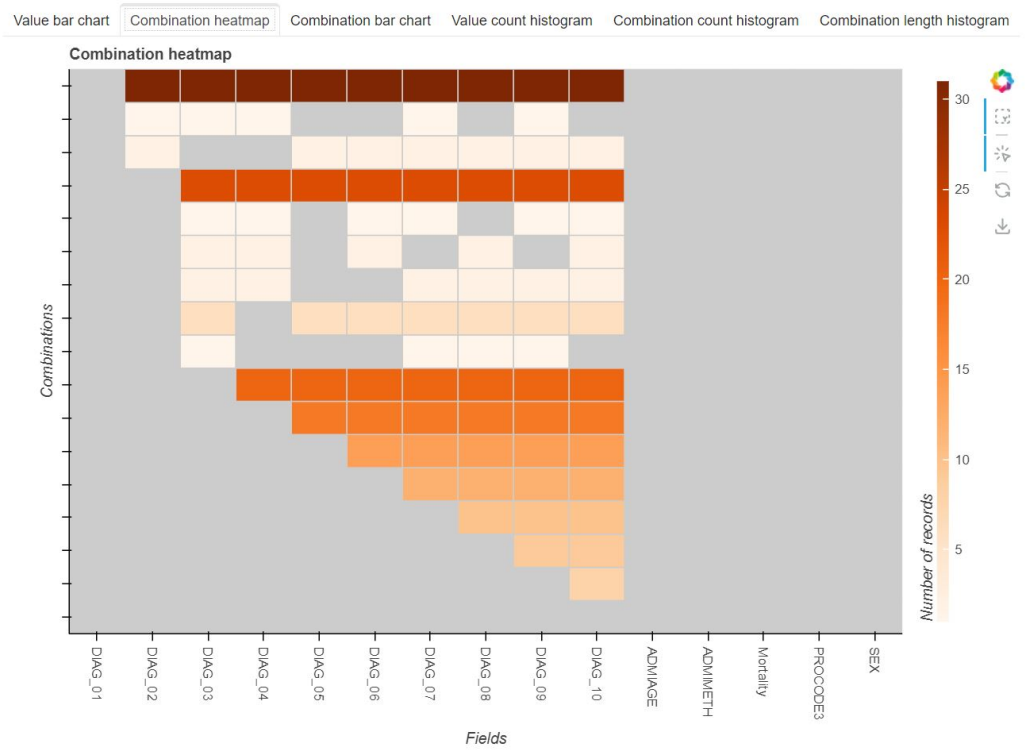

Figure 1: An example APC combination heatmap shows the fields (X axis), each combination of missing values (Y axis) and the number of records that are in each combination (colour) of the APC (Admitted Patient Care) dataset included in the package. The top, 4th from top and bottom six combinations are a monotone pattern. However, the other seven combinations show that there is another pattern that has gaps in the DIAG fields.

SetVis provides the same six built-in visualizations as ACE (Ruddle, Adnan, Kavanagh, et al., 2022). The main two show visualizations of sets (in a bar chart) and set intersections (in a heatmap). The other four visualizations make SetVis scalable to data that contains large numbers of sets and/or intersections, by showing histograms of set cardinality, intersection degree and intersection cardinality, and an intersection degree bar chart. All of the visualizations are interactive (implemented with Bokeh, Bokeh Development Team, 2018), but users may also interact programmatically and freely interleave the two forms of interaction. Examples and tutorials are provided with the installation. A screenshot of SetVis version `v0.1rc5` of the heatmap visualizations within Jupyter notebooks is shown in Figure 1.

Jupyter notebooks have been widely adopted in the Python data science ecosystem for exploratory data analysis. It is considered good practice for computational notebooks to obey principles of (i) top-to-bottom re-executability and (ii) repeatability, including by others (Quaranta et al., 2022). The SetVis design allows these principles to be respected.

SetVis is underpinned by memory-efficient data structures. Set membership information for each of set membership information for each of K sets can be represented with a mapping from an element (represented by its index) to a tuple of K booleans based on indicator functions for each of these sets:

$$members : ElementIndex \rightarrow \{True, False\}^K. \quad (1)$$

One component of the resulting tuple indicates membership of a particular set. Storing this mapping explicitly (e.g., as in UpSet with a dataframe, Conway et al., 2017; Nothman, 2022) requires $O(KN)$ storage, where K and N are the number of sets and the number of elements. When K is large, as is the case for many real-world datasets, this can be inefficient. The number of unique set intersections, R, is often much smaller than the number of records, $R \ll N$, and can be at most N (if each element is member of a unique combination of sets). SetVis makes use of this idea, and considers

$$members = intersectionMembers \circ intersectionId \tag{2}$$

where

$$intersectionId : ElementIndex \rightarrow IntersectionIndex \tag{3}$$

maps an element index to an index referring to the particular combination of sets to which that element belongs; and

$$intersectionMembers : IntersectionIndex \rightarrow \{True, False\}^K \tag{4}$$

is a bijection between an intersection index and the explicit representation of this combination.

In SetVis, these mappings are stored as a pair of Pandas dataframes (in an instance of the *Membership* class), $intersectionId$ of size $O(N)$ and $intersectionMembers$ of size $O(RK)$, for a combined total of $O(N + RK)$ storage.

Technical evaluation

Using a Ubuntu virtual machine with 44GB of RAM, we compared SetVis (v0.1.0) with UpSetPlot (v0.8.0) based on two criteria: memory use and compute time. The greatest difference was in memory usage. The UpSetR package (Conway et al., 2017) was not tested, but uses a similar data structure to UpSetPlot (Nothman, 2022).

Tests were run with set-type data that contained two columns, 500,000 rows and 100 – 10,000 set intersections. UpSetPlot crashed when the 500,000 row dataset contained more than 500 intersections. By contrast, SetVis only used 113 MB RAM for 500,000 rows with the maximum 10,000 set intersections (see Figure 2A).

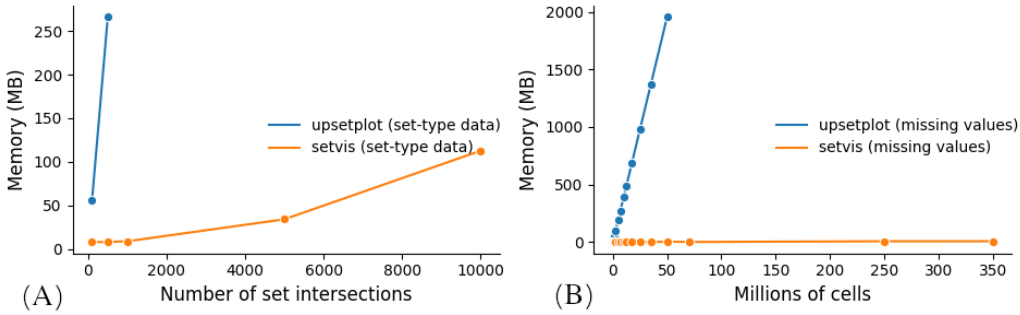

Figure 2: (A) Memory used by UpSetPlot and SetVis for set-type data with 500,000 rows, two columns and a range of set intersections. There were always 10% more sets than intersections. (B) Memory used by UpSetPlot and SetVis for visualizing patterns of missing data. The number of cells equals the number of rows × columns in a dataset.

The difference was even more pronounced when the packages were used to analyze missing data (10,000 – 500,000 rows; 10 – 700 columns; each row missing one value). UpSetPlot's memory scaled linearly with the number of cells (i.e., rows × columns) in a dataset, whereas SetVis's memory only increased gradually (see Figure 2B). There was a similarly large difference when

each row contained 1 – 50 missing values (100 – 10,000 set intersections in each dataset), because for missing data UpSetPlot keeps a copy of the input Pandas dataframe, as well as having a memory-hungry design.

Acknowledgements

This research was supported by the Alan Turing Institute and the Engineering and Physical Sciences Research Council (EP/N013980/1; EP/R511717/1).

References

Adnan, M., & Ruddle, R. (2018). A set-based visual analytics approach to analyze retail data. *Proceedings of the EuroVis Workshop on Visual Analytics (EuroVA18)*. https://doi.org/10.2312/eurova.20181110

Alper, B., Riche, N., Ramos, G., & Czerwinski, M. (2011). Design study of LineSets, a novel set visualization technique. *IEEE Transactions on Visualization and Computer Graphics*, *17*(12), 2259–2267. https://doi.org/10.1109/TVCG.2011.186

Alsallakh, B., & Ren, L. (2016). Powerset: A comprehensive visualization of set intersections. *IEEE Transactions on Visualization and Computer Graphics*, *23*(1), 361–370. https://doi.org/10.1109/TVCG.2016.2598496

Bokeh Development Team. (2018). *Bokeh: Python library for interactive visualization*. https://bokeh.pydata.org/en/latest/

Conway, J. R., Lex, A., & Gehlenborg, N. (2017). UpSetR: An R package for the visualization of intersecting sets and their properties. *Bioinformatics*, *33*(18), 2938–2940. https://doi.org/10.1093/bioinformatics/btx364

Dörk, M., Riche, N. H., Ramos, G., & Dumais, S. (2012). Pivotpaths: Strolling through faceted information spaces. *IEEE Transactions on Visualization and Computer Graphics*, *18*(12), 2709–2718. https://doi.org/10.1109/TVCG.2012.252

Freiler, W., Matkovic, K., & Hauser, H. (2008). Interactive visual analysis of set-typed data. *IEEE Transactions on Visualization and Computer Graphics*, *14*(6). https://doi.org/10.1109/TVCG.2008.144

Jia, A., Xu, L., & Wang, Y. (2021). Venn diagrams in bioinformatics. *Briefings in Bioinformatics*, *22*(5), 1–17. https://doi.org/10.1093/bib/bbab108

Kim, B., Lee, B., & Seo, J. (2007). Visualizing set concordance with permutation matrices and fan diagrams. *Interacting with Computers*, *19*(5-6), 630–643. https://doi.org/10.1016/j.intcom.2007.05.004

Lamy, J.-B., & Tsopra, R. (2019). RainBio: Proportional visualization of large sets in biology. *IEEE Transactions on Visualization and Computer Graphics*, *26*(11), 3285–3298. https://doi.org/10.1109/TVCG.2019.2921544

Landolfi, A., Picillo, M., Pellecchia, M. T., Troisi, J., Amboni, M., Barone, P., & Erro, R. (2022). Screening performances of an 8-item UPSIT Italian version in the diagnosis of Parkinson's disease. *Neurological Sciences*, 1–7. https://doi.org/10.1007/s10072-022-06457-2

Lex, A., Gehlenborg, N., Strobelt, H., Vuillemot, R., & Pfister, H. (2014). UpSet: Visualization of intersecting sets. *IEEE Transactions on Visualization and Computer Graphics*, *20*(12), 1983–1992. https://doi.org/10.1109/TVCG.2014.2346248

Molbiotools. (2022). *MOLBIOTOOLS - Molecular Biology Online Apps*. https://molbiotools.com/

Nothman, J. (2022). *UpSetPlot*. https://pypi.org/project/UpSetPlot/

Quaranta, L., Calefato, F., & Lanubile, F. (2022). Eliciting best practices for collaboration with computational notebooks. *Proceedings of the ACM on Human-Computer Interaction*, *6*(CSCW1), 1–41. https://doi.org/10.1145/3512934

Ruddle, R., Adnan, M., & Hall, M. (2022). Using set visualisation to find and explain patterns of missing values: A case study with NHS hospital episode statistics data. *BMJ Open*, *12*(11), e064887. https://doi.org/10.1136/bmjopen-2022-064887

Ruddle, R., Adnan, M., Kavanagh, R., Strickson, O., & Wochner, P. (2022). *The ACE software, and training materials for visualizing missing data and set-type data*. https://doi.org/10.5518/1150

Wang, Q., Xu, Z., Chen, Z., Wang, Y., Liu, S., & Qu, H. (2020). Visual analysis of discrimination in machine learning. *IEEE Transactions on Visualization and Computer Graphics*, *27*(2), 1470–1480. https://doi.org/10.1109/TVCG.2020.3030471

Yalcin, M. A., Elmqvist, N., & Bederson, B. B. (2015). AggreSet: Rich and scalable set exploration using visualizations of element aggregations. *IEEE Transactions on Visualization and Computer Graphics*, *22*(1), 688–697. https://doi.org/10.1109/TVCG.2015.2467051