Spreadsheet Data Extractor (SDE): A Performance-Optimized, User-Centric Tool for Transforming Semi-Structured Excel Spreadsheets into Relational Data

ANONYMOUS AUTHOR(S)
SUBMISSION ID: 1505

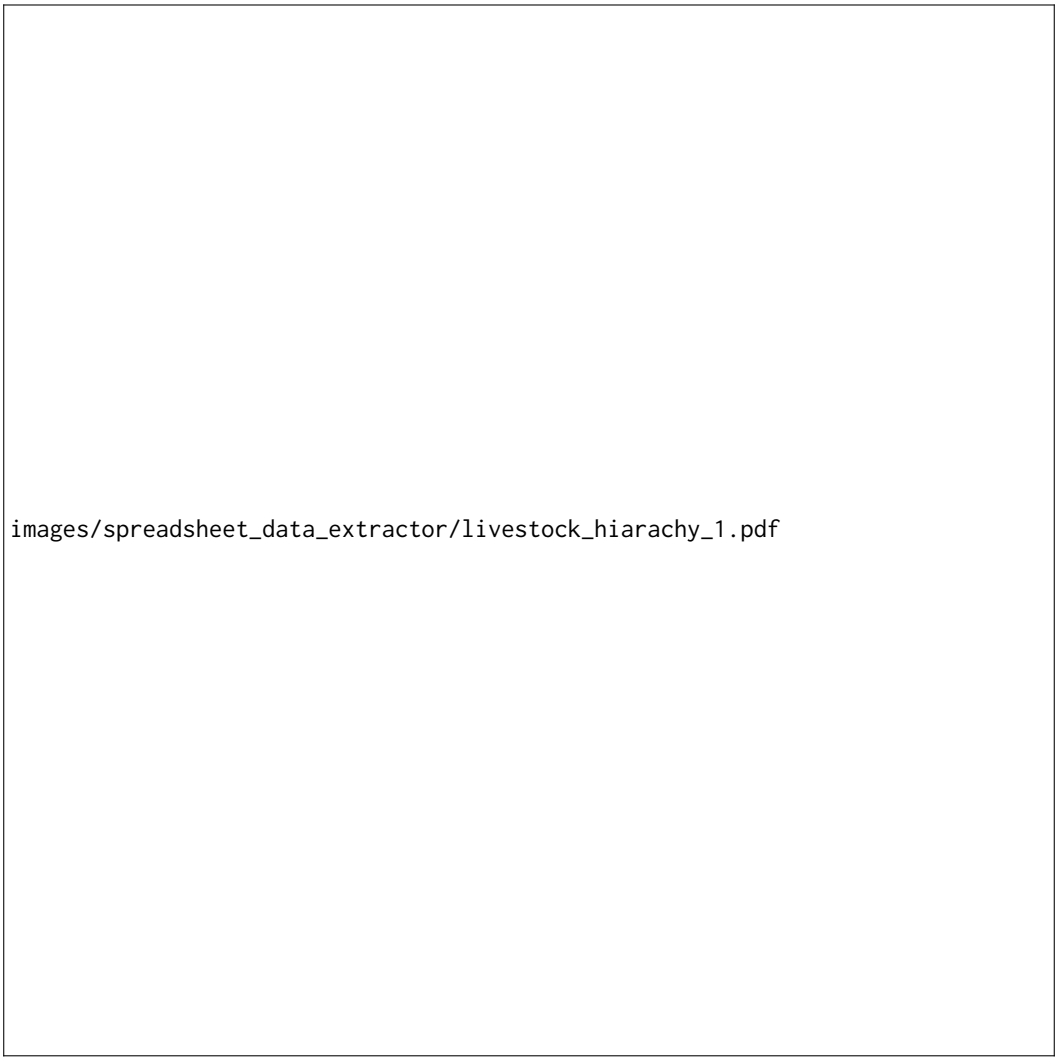


Fig. 1

CCS Concepts: • **Applied computing** → **Spreadsheets**; • **Information systems** → **Data cleaning**; • **Software and its engineering** → **Extensible Markup Language (XML)**; • **Human-centered computing** → *Graphical user interfaces*; • **Theory of computation** → Data compression.

ACM Reference Format:
Anonymous Author(s). 2025. Spreadsheet Data Extractor (SDE): A Performance-Optimized, User-Centric Tool for Transforming Semi-Structured Excel Spreadsheets into Relational Data. In *Proceedings of 2025 International Conference on Management of Data (SIGMOD '25).* ACM, New York, NY, USA, **??** pages. https://doi.org/10.1145/XXXXXXX.XXXXXXX

1 INTRODUCTION

[**?**] A multitude of organizations across various sectors, including healthcare[**?**], nonprofit[**?**], [**?**], finance, commerce, industry, academia, and government [**?**] frequently encounter difficulties when attempting to analyse and utilize semi-structured data derived from spreadsheets. The absence of a defined structure within this data presents significant challenges for analysis and utilization.

Nevertheless, it is not always the case that there is no structure at all. There are cases where the data at least has "a" structure. However, this structure is almost as helpful as if there were simply none at all. This is the case for data stored in spreadsheet files like Excel. The data is not stored in a simple machine-readable wide- or long-format; rather, it is stored in a format that is designed for human perception, including layout with empty cells for padding, combined cells, hierarchies in column headers, multiple tables below one another, and so on. This is illustrated in Figure **??** These layout techniques facilitate the comprehension of the information by humans, but they also render the data non-machine readable. A computer program treats the spreadsheet as a two-dimensional array of values. It treats empty cells in the same way as it treats cells that contain actual data.

Compared to unstructured data in raw text, unstructured data in spreadsheets has the following advantages: The hierarchy of metadata within spreadsheets is easy to comprehend for a human. Headlines like *Germany* and *Berlin*, referring to our example in Figure **??**, combined column headers like *Stock of Calves and Young Cattle* on top of multiple other column headers like *Quantity* and *LU* make the metadata hierarchy clear to humans. The target data structure can be closely inspired by that visible hierarchy and can be created with a relatively low amount of domain knowledge. Furthermore, the values are encoded in single cells and can be uniquely referenced by the file, sheet name and row and column index.

The objective is to transform this data into a machine-readable form. When considering the means to achieve this objective, several approaches present themselves.

The SDE converts semi-structured spreadsheet data into structured formats by allowing users to define data hierarchies without programming knowledge.

The Spreadsheet Data Extractor (SDE) enables users to transform the data from spreadsheets into structured data by selecting cells containing metadata and values to describe the data hierarchy. To streamline the process, users can copy previously described hierarchies to similar spaces in the sheets, eliminating the need to select every cell and hierarchy repeatedly. Users do not require any programming knowledge to extract the data from the spreadsheet files.

SIGMOD '25, June 22–27, 2025, Berlin, Germany

ACM ISBN 978-1-4503-XXXX-X/25/06…$15.00
https://doi.org/10.1145/XXXXXXX.XXXXXXX

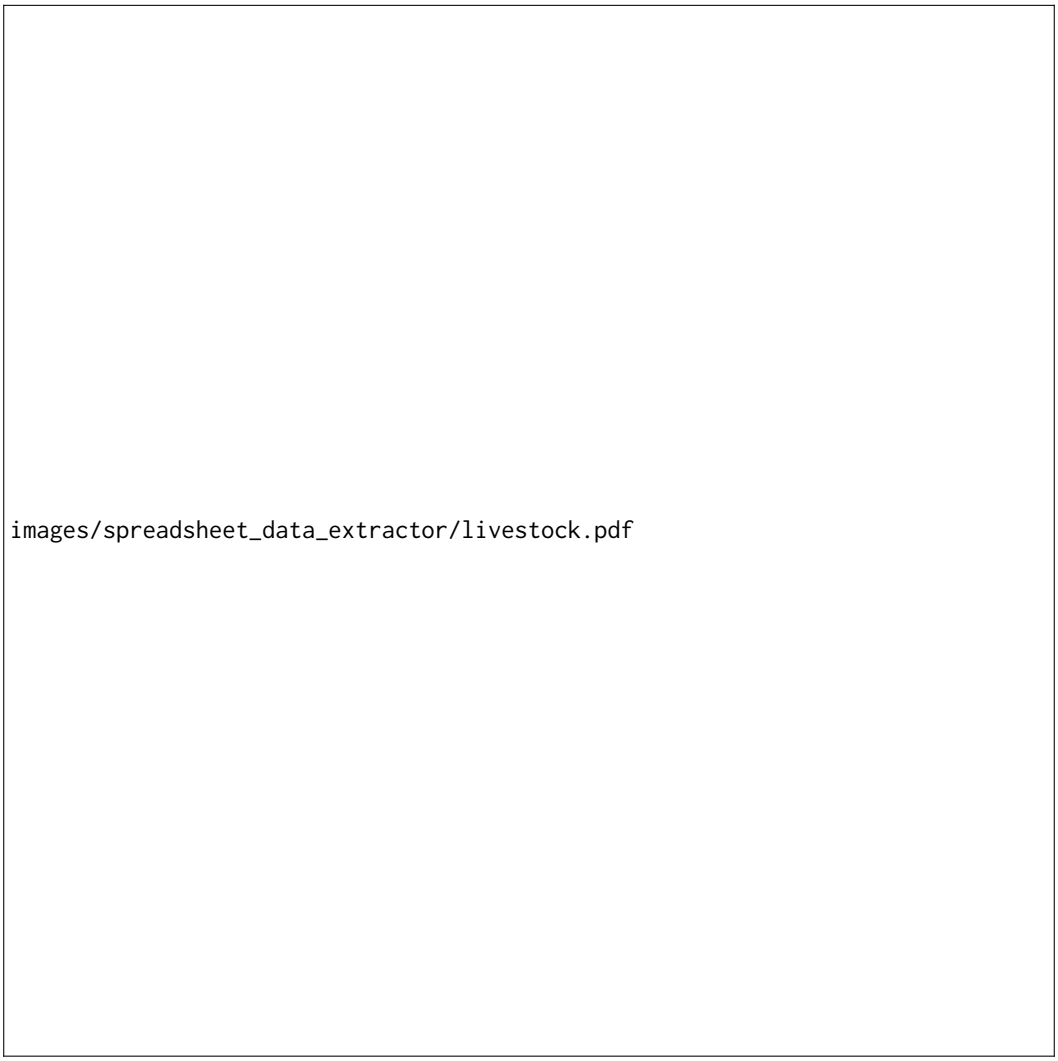


Fig. 2. Example data subset of the Agricultural Structure Survey on land use and livestock in Germany, Source: Own figure

2 RELATED WORK

This chapter will first describe the similar approaches of data extraction from spreadsheets without programming that are found in scientific literature.

3 SPREADSHEET DATA EXTRACTOR

Our program's closest research counterparts are the projects, Senbazuru, and FlashRelate, both of which share the goal of extracting relational data from spreadsheets with a graphical user Interface. [?],[?],[?]

Name	Technologies	Output	Accessibility	Frequency
FlashRelate	AI, GUI	cleaned data	proprietary, no access	last publication 2015
Senbazuru	AI, GUI	cleaned data	open source	last commit 2015
XLindy	AI	cleaned data	no access	cancelled
TableSense	AI	diagrams	proprietary, no access	last commit 2021

Table 1. Spreadsheet Data Extractor counterparts

Senbazuru aims to automatically infer the hierarchical structure within spreadsheets through the creation of a classifier that identifies data frames in the document, along with another classifier that infers the intended hierarchy using a predefined set of features [**?**].

While Senbazuru offers automated inference of the hierarchical structure, it may occasionally produce errors. In such cases, the tool allows the user to rectify these errors in the hierarchy by manually dragging and dropping nodes to their correct parent locations. Unlike Senbazuru, our program does not automatically infer the hierarchical structure. Instead, our program lets the user decide if the next selection is a child or a sibling of another selection. To speed up the process, selections can be duplicated and moved both horizontally and vertically. Thus, a hierarchy representing a column can be duplicated and shifted over another column, and similarly, a hierarchy representing a table can be duplicated and shifted over another table.

FlashRelate utilizes both positive and negative output examples to generate a program in a domain-specific language called Flare.[**?**] This approach enables FlashRelate to handle various extraction tasks from diverse spreadsheets, as long as it is possible to identify cells using regular expressions and recognize the spatial arrangement of the cells. In contrast to FlashRelate, our program necessitates the user to manually choose each data cell containing either data or metadata. It does not automatically identify similar cells based on user-provided selections. Instead, our program allows users to select not only individual cells but also sets of cells. Portions of the spreadsheet that represent relational data rather than hierarchical data can be chosen by first selecting the domain of the relation as a set of cells. Typically, the range values of the relation share the same row as the domain values. In a subsequent step, the ranges of the relation can be selected by duplicating the cell set of the domain and moving it horizontally to the desired cells containing the range values.

There are also two projects that rely on Artificial Intelligence only to identify the hierarchy of the data inside spreadsheets. Those projects are XLindy and TableSense. [**?**],[**?**]

XLindy is not in development any more. Attempts were made to contact the authors to try the tool and find out if using it on our test dataset results in reliable output. Due to the fact that the project is no longer in development the tool could not be obtained and tested.

TableSense employs Convolutional Neural Networks to identify tables in spreadsheets.[**?**] In their test with the WebSheet400 dataset, out of 400 random spreadsheets containing 795 tables, TableSense successfully detected 91.3% of the tables (recall), and of the tables detected by TableSense, 86.5% were indeed genuine tables (precision). However, while TableSense excels at detecting table boundaries, it does not identify data hierarchy or extract data from the tables it identifies. Our approach currently does not rely on Artificial Intelligence. Our project's primary focus is to empower users to manually select all the desired data and describe its hierarchy, enabling our program to automatically extract this data into a relational format.

4 METHODOLOGY

We used the software development kit Flutter to develop a program with a user interface that allows the selection of the cells containing the metadata and data inside the spreadsheet. We refer again to the example from Figure **??** to describe the necessary steps required for data extraction from spreadsheets. We use this example spreadsheet to illustrate the necessary steps required for data extraction from spreadsheets like this one. When looking at the spreadsheet, a human can identify a hierarchy very easily. There are sub-tables with headings, and all the sub-tables share the same columns, which are only laid out once at the top of the spreadsheet rather than being repeated for each sub-table.

The headings *Germany* and *Berlin* in this example are the topmost metadata because most cells containing actual data share these headings as common metadata, whereas the metadata of the column header *Agric. Utilised Area* and *Hectares ha* is only shared by the cells beneath that column. A human can see these hierarchies, but to a program, these hierarchies are not visible; the program just sees a two-dimensional grid of cells, some empty, some containing text, and some containing numbers.

The computer needs to be able to interpret this hierarchy in the spreadsheet just as a human does. However, for that to happen, it needs the hierarchy as input. Our approach is that the user provides this hierarchy to the computer by selecting the cells and describing the relationships they have to one another.

To make this process easier and faster for the user, the amount of user interaction should be minimized as much as possible. This can be achieved by selecting the most abstract meta-information first and then progressively narrowing down to increasingly specific meta-information until the actual facts are chosen. Additionally, allowing the user to duplicate selections and hierarchies to similar areas in the spreadsheet can streamline this process.

In Figure **??**, cells are presented as they appear within our program's data selection view after all the selections are made. Additionally, an additional layer with black arrows illustrates the optimal sequence of selections for this example, designed to minimize user interaction. These arrows, numbered from 1 to 7, represent the specific steps of user interaction.

Please note that the program deconstructs merged cells. Cells from which the merged cell was created are displayed as individual cells (e.g. *Quantity* and *LU*). This will be useful for the following selections.

The most abstract meta-information, which is the meta-information shared by most cells, is the country or state in this example (Arrow 1). After that, the header of the domain of the relation is selected (Arrow 2) and then the domain itself will be picked (Arrow 3). While Selections 1 and 2 each selected individual cells, Selection 3 introduces the concept of choosing a set of cells. Moving on to the first data column, the process involves selecting the first column header (Arrow 4), followed by choosing the second column header (Arrow 5). Now, the first range of the relation is selected (Arrow 6). If the selection of data were to continue in the same manner, this would involve an additional 15 selections just for the first table, offering no significant improvement over manually creating the relation by simply copying and pasting the cells. However, the subsequent selections can be automated, given that both two column headers and the numeric values are located in the same row, each one cell to the right of the previous one. The program's *Duplicate and Move* function allows these cells to be selected in a single step. To do this, Selection 4 is chosen, and the duplicate is moved one cell to the right.

A slider can be adjusted to determine the number of duplicates to generate. Each duplicate is moved the same number of cells in the same direction as the previous one. For this example, the

slider is set to create 5 duplicates (see Figure **??**). This results in all the selections depicted as arrows labelled 7 in Figure **??**.

This functionality is made possible by the Spreadsheet Data Extractor's ability to deconstruct merged cells, such as those in the *Quantity* and *LU* columns. Without this feature, duplicating and moving selections over these merged columns would fail, resulting in the cells within the *Quantity* and *LU* columns remaining empty, except for the first cell in each merged group. However, the Spreadsheet Data Extractor displays the underlying cells of the merged cells as redundant copies. This approach allows the process of duplicating and moving column selections over these merged areas to work seamlessly. With this, the first table is captured.

The same procedure used to capture the previous five columns can be applied to capture the tables arranged below one another. To do this, Selection 1 is chosen, then duplicated and moved down by six cells. However, the header cells (B1 to H2) are an exception to this, because they don't reappear in the tables below. The program offers a freeze option for this purpose. The frozen cells are duplicated but remain in their original positions during the move. Figure **??** illustrates how the *Duplicate and Move* function was employed to capture the table below by moving all but the frozen cells down by six cells (all arrows labelled with the number 8).

After the selections are made, the hierarchy of selections forms a tree in which each node contains the selected set of cells. Except for the first two nodes, where the first node represents the spreadsheet file (e.g. *Livestock.xlsx*) and the second node represents the workbook's worksheet (e.g. *Calves*). This is demonstrated in Figure **??** for the subtree with the root node *Germany*.

To transform this tree of sets of cells into relational data, we calculate the cross product of the tree. The length of each child set is obtained from the bottom up, and the parent node is duplicated as many times as necessary to match the length of that child set (Figure **??**).

If the length of the parent and child nodes differ and the length of the parent set is not a whole number divisor of the length of the child set, an error will be thrown, as it is not possible to duplicate the parent set to match the length of the child set. This situation should not occur. When a parent set consists of more than one cell, it is usually because the parent set represents the identifier of the rows, or in mathematical terms, the domain of the relation. This domain must be of the same length as the values represented to the right of the rows' identifier, which correspond to the relation range in mathematical terms. This is also applicable to the table in Figure **??**, where the relation's domain comprises the cells *1-9*, *10-19*, and *20-49*, while the range is located to the right of it.

The final step is to convert each list of sets to tuples and stack them on top of each other to create the final relation, as shown in **??**. This relation can then be exported as a CSV file.

4.1 Serialisation

4.2 Performance

The amount of unsued cells is likely to be higher, because our script does not check for cells that seem like they contain contentbecause they reference a shared structuring but in reality those shared strings are just whitespaces that got into the cells by accident. Those cells are not filter our by our script because of the performance. Our script filters out the cells that don't have childdren with a xpath query which is very fast in comparison to iterating over every node and checking if the referenced shared string consists of only white spaces.

https://ecma-international.org/publications-and-standards/standards/ecma-376/

Dataset: [**?**]

5 EVALUATION

After building the first version of the Spreadsheet Data Extractor, student assistants successfully used it to extract data from over 500 Excel files. The time taken for each file was determined from a sample of 331 processed Excel files, comprising 3,093 worksheets, with outliers caused by pauses removed. On average, the student assistants required 15 minutes per file and 95 seconds per worksheet.

Research Question 1 can be answered positively. Student assistants without programming experience were able to extract data from over 500 Excel files using the Spreadsheet Data Extractor in a reasonable amount of time.

6 OUTLOOK

The extraction of data from the heterogeneous spreadsheet files was possible, even though the user experience was not optimal. Many aspects of the user experience that would have been significant time-saving features had to be omitted during development due to time restrictions. These included features for describing the hierarchy of the worksheets and exporting the extracted data.

6.1 Improvements for Describing the Spreadsheet Hierarchy

The current version of the Spreadsheet Data Extractor hampers the user experience:

- The user needs to switch back and forth between the task view (Figure **??**) and the cell selection view (Figure **??**).
- To duplicate and move a hierarchy, the user must first select each node that should not be moved individually and then freeze them in place.
- The user needs to know in advance when to create empty nodes if the number of column headers varies from column to column.

6.1.1 Switch Between Task View and Cell Selection View. When creating a new node to describe the task of extracting a cell or set of cells, the user must switch to the view that displays all tasks. To export a cell or set of cells, the user must first switch to the view that displays the cells and then select the desired cell(s) upon node creation. As the hierarchy of the spreadsheet grows, switching back and forth and finding the correct nodes becomes increasingly time-consuming.

6.1.2 Select Each Node Individually to Lock in Place. The switching of views is increasingly complicated for the user when duplicating and moving a hierarchy of export tasks. While in the view showing the cells, the user might try to duplicate a hierarchy by extracting one sub-table of a sheet and then move that hierarchy to the next sub-table below. Only at that point, the user might see that the table below does not repeat the column headers, and the user might want to freeze those nodes in place.

To freeze the nodes the user needs to switch to the task view once again. There, they need to look up all nodes that represent the column headers. The user can only identify those nodes by their extracted value and their cell coordinate, while it would be much easier to just see the centre in the actual grid. The column headers are scattered across the whole hierarchy, which means it will take the user a while to scroll through the hierarchy, reminding themselves of the extracted values from those column headers and comparing them with the values shown in the task list.

After the user has used the freeze function, they can switch back to the view showing the worksheet cells and repeat the step of duplicating and moving the hierarchy. Only by that time will the user see if they correctly froze the correct nodes or if they missed nodes or even selected the wrong ones.

6.1.3 The User Needs to Know Beforehand When to Create Empty Nodes. Many tables in our dataset have columns with varying count of headers. For instance, in Figure **??** [**?**], there is one column with the header *Total* and another with the headers *Ewes*, *Dairy Sheep*, and *Namely*. The header *Namely* is unnecessary, but *Ewes* is valuable metadata and should be included in the output relation. To preserve all metadata, there are two possible solutions. Columns with multiple headers could be concatenated with a delimiter, such as *Ewes-Dairy Sheep*. This would ensure that the number of headers in the column is consistent. However, this solution has the disadvantage of losing the hierarchy of the headers, making it impossible to filter the resulting dataset for the metadata *Ewes*. The second solution would be to add empty nodes for the columns with fewer headers to match the count of headers in all columns. In this example, *Total* and *Dairy Sheep* can be treated as metadata of the same level.

Adding an empty node with the child *Total* results in the output relation for the rows *Total* and *Dairy Sheep*, as shown in Figure **??**.

To accomplish this, the user must be aware at the moment they select the *Total* column, that subsequent columns will have additional column headers. Therefore, they need to preemptively create an empty node as the parent of the *Total* node. If the user forgets to create this parent node and only realizes the discrepancy in the number of column headers in subsequent columns, they currently have to tediously modify the hierarchies of the previous columns. The user can initially copy the *Total* node, then delete it, add an empty node in its place, and finally insert the copied *Total* node as its child. This process involves a significant number of steps for the user, resulting in a poor user experience.

Improving the user experience of this application could be achieved through several enhancements:

6.1.4 Integration of Task and Cell Selection Views. The hierarchy of cell selections could be shown as a side panel, or even better as graphs in the view showing the cells. In the current version of the Spreadsheet Data Extractor the selected cells are already highlighted. The hierarchy could be shown by connecting the highlighted sets of cells with arrows pointing to their parent and child nodes.

6.1.5 Support for Multi-Node Selection. If the user wishes to freeze cells before duplicating and moving a hierarchy, the user should be able to select these cells by clicking directly on them. This could be sped up by allowing the user to select multiple sets of cells by drawing a line around them, as is done in other GUIs and is often referred to as the lasso tool.

6.1.6 Drag-and-Drop Node Manipulation. Enabling users to drag and drop nodes to rearrange them within the hierarchy would simplify the process of organizing and structuring data, providing a more intuitive way to describe the hierarchy and allowing for quick corrections in case of errors.

6.1.7 Node Wrapping Functionality. Introducing the feature to wrap user-selected nodes into new ones would provide users with the flexibility to create empty nodes on-the-fly as they recognize the need for them. For example, if a user adds a new column with more column headers than the existing ones, they could seamlessly wrap the previous column headers into new nodes to match the number of headers in each column.

6.2 Enhancements for Data Export

While the extraction of the more than 500 Excel files was a success, it resulted in a few hundred CSV files. All of these files are in a machine-readable format but are split into many separate files. A lot of those files have the same amount columns and the same or a very similar set of metadata columns. Those files should be combined, because they contain the same type of data, yet different

values. The reason for this is, because the Spreadsheet Data Extractor made it easy for the user to extract data from just one or a few of Excel files. Using it for a lot of files was impractical, because the task view grew very big with just extracting from one file. To maintain an overview the student assistants instead created a new configuration when moving to the next Excel file. That means that the extracted files needed to be combined afterward by comparing them with each other. When two files are similar, they needed to be merged into a single file

To streamline this process the user experience could be improved by implementing an input- and output-manager.

6.2.1 Input Manager. Firstly, an Input Manager within the Spreadsheet Data Extractor should facilitate the import of all files, offering users a comprehensive overview of all imported files directly within the application. This feature would discourage users from organizing files externally and ensure centralized management of file imports.

Additionally, the Input Manager should indicate the extent to which each file's data has been described through cell selections. This would aid users in finding out which files need to be processed next and prevent the need for manual organization of files into directories called "to-do" and "done".

6.2.2 Output Manager. Secondly, the Spreadsheet Data Extractor requires an Output Manager. Instead of exporting hierarchies directly to CSV files, users should first define a target format specifying column names, data types, and desired order. Subsequently, users can progressively link newly described hierarchies to existing output formats. This facilitates efficient data consolidation. Once all files are described and linked, data can be exported as a small set of CSV files, ensuring consistency and ease of use.

REFERENCES

[] Daniel W Barowy, Sumit Gulwani, Ted Hart, and Benjamin Zorn. 2015. FlashRelate: extracting relational data from semi-structured spreadsheets using examples. *ACM SIGPLAN Notices* 50, 6 (2015), 218–228.

[] D.J. Berndt, J.W. Fisher, A.R. Hevner, and J. Studnicki. 2001. Healthcare data warehousing and quality assurance. *Computer* 34, 12 (2001), 56–65. https://doi.org/10.1109/2.970578

[] Zhe Chen, Michael Cafarella, Jun Chen, Daniel Prevo, and Junfeng Zhuang. 2013. Senbazuru: A prototype spreadsheet database management system. *Proceedings of the VLDB Endowment* 6, 12, 1202–1205.

[] Haoyu Dong, Shijie Liu, Shi Han, Zhouyu Fu, and Dongmei Zhang. 2019. Tablesense: Spreadsheet table detection with convolutional neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 69–76.

[] Angus Dunn. 2010. Spreadsheets-the Good, the Bad and the Downright Ugly. *arXiv preprint arXiv:1009.5705* (2010).

[] Elvis Koci, Dana Kuban, Nico Luettig, Dominik Olwig, Maik Thiele, Julius Gonsior, Wolfgang Lehner, and Oscar Romero. 2019. Xlindy: Interactive recognition and information extraction in spreadsheets. In *Proceedings of the ACM Symposium on Document Engineering 2019*. 1–4.

[] R Core Team. 2019. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

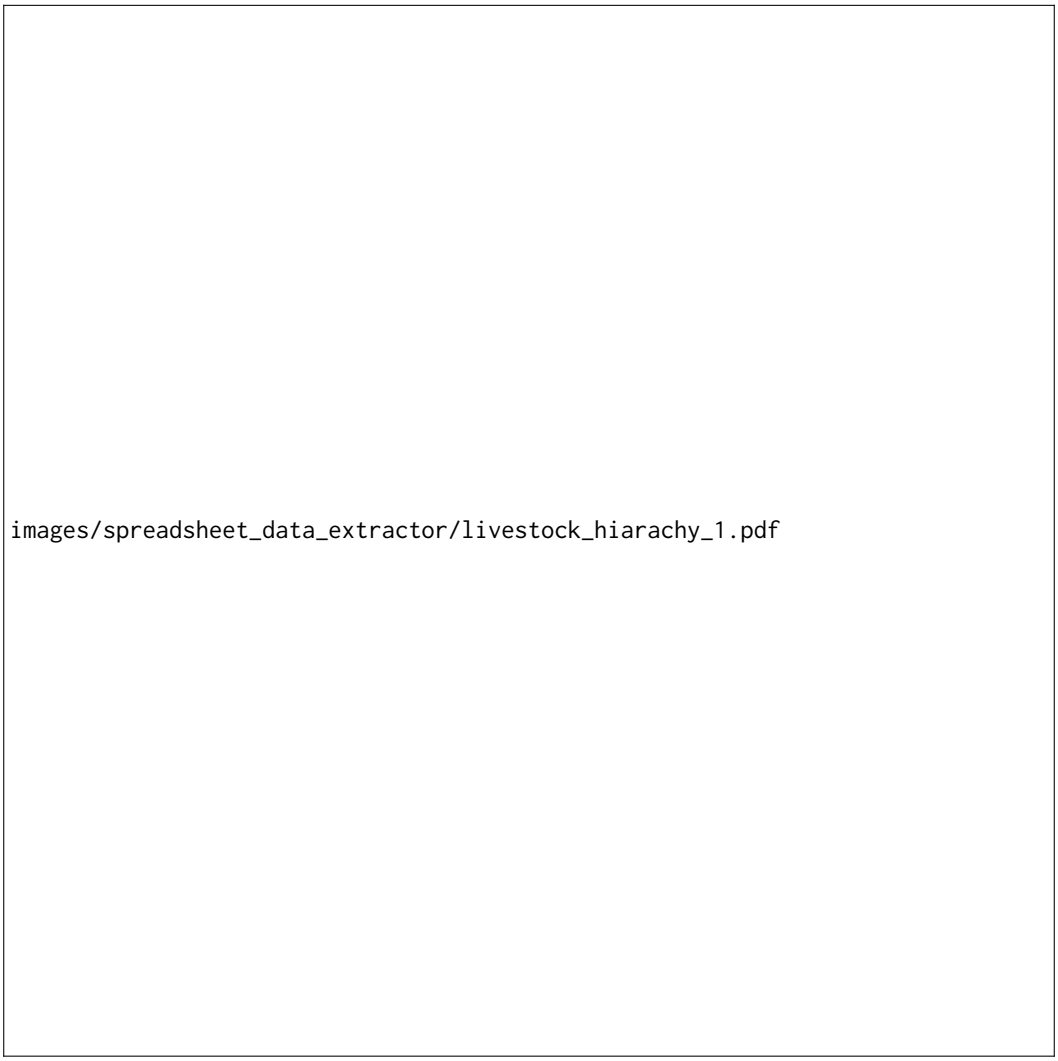
[] Gursharan Singh, Leah Findlater, Kentaro Toyama, Scott Helmer, Rikin Gandhi, and Ravin Balakrishnan. 2009. Numeric paper forms for NGOs. In *2009 International Conference on Information and Communication Technologies and Development (ICTD)*. IEEE, 406–416.

[] Statistisches Bundesamt. [n.d.]. Viehhaltung der Betriebe - Fachserie 3 Reihe 2.1.3 - 2020 (Letzte Ausgabe - berichtsweise eingestellt). Online. https://www.destatis.de/DE/Themen/Branchen-Unternehmen/Landwirtschaft-Forstwirtschaft-Fischerei/Tiere-Tierische-Erzeugung/Publikationen/Downloads-Tiere-und-tierische-Erzeugung/viehhaltung-2030213209005.xlsx, accessed 30 Oct 2023.

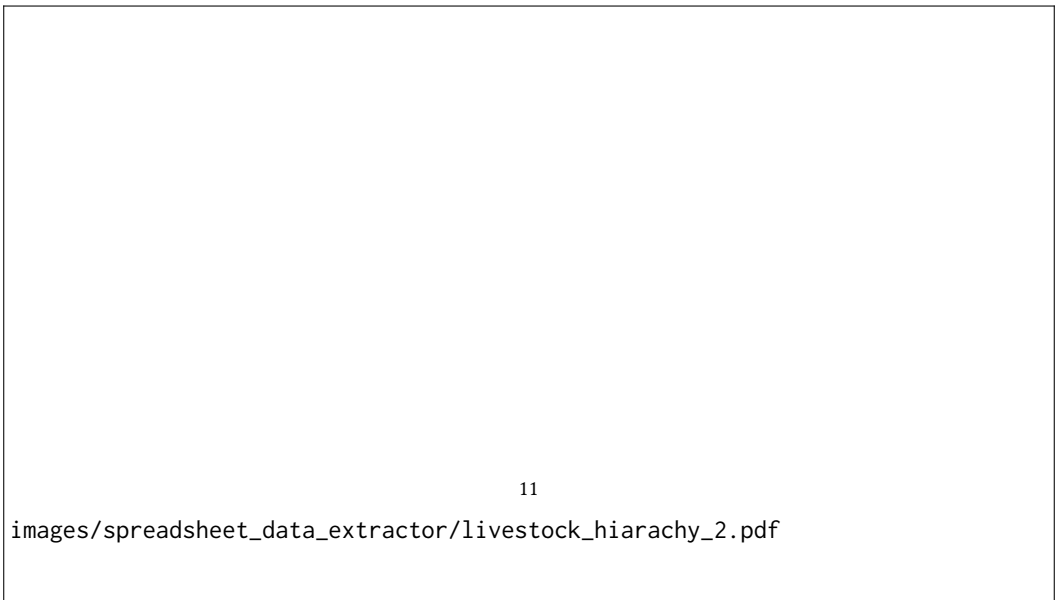
[] C. Vos, C. Rösemann, H.-D. Haenel, U. Dämmgen, U. Döring, S. Wulf, B. Eurich-Menden, A. Freibauer, H. Döhler, B. Steuer, B. Osterburg, and R. Fuß. 2024. *Calculations of Gaseous and Particulate Emissions from German Agriculture 1990 - 2022: Input Data and Emission Results*. https://doi.org/10.3220/DATA20240219164429-0

[] Hannah West and Gina Green. 2008. Because excel will mind me! the state of constituent data management in small nonprofit organizations. In *Proceedings of the Fourteenth Americas Conference on Information Systems*. Association for Information Systems, AIS Electronic Library (AISeL). https://aisel.aisnet.org/amcis2008/336

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009



(a) The data selection view, once all selections are completed



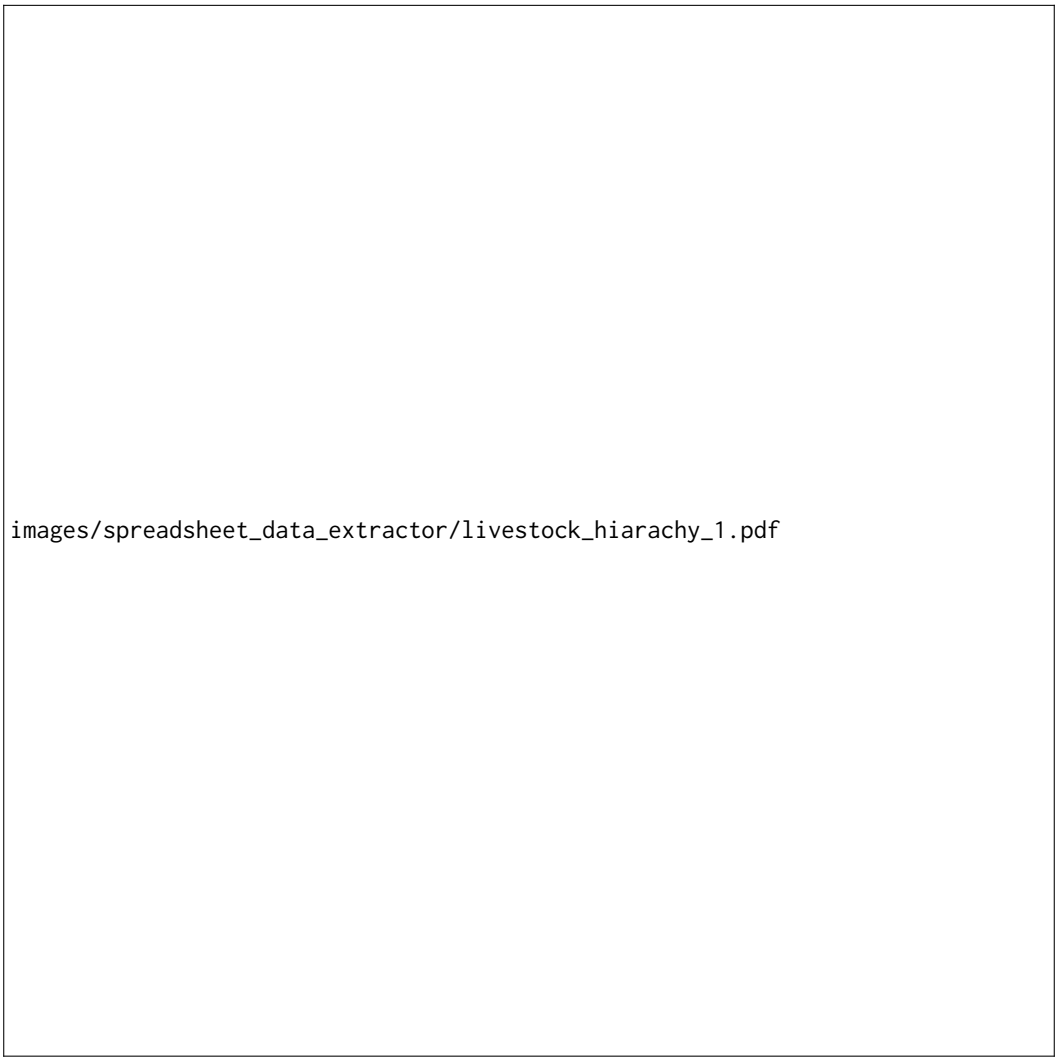
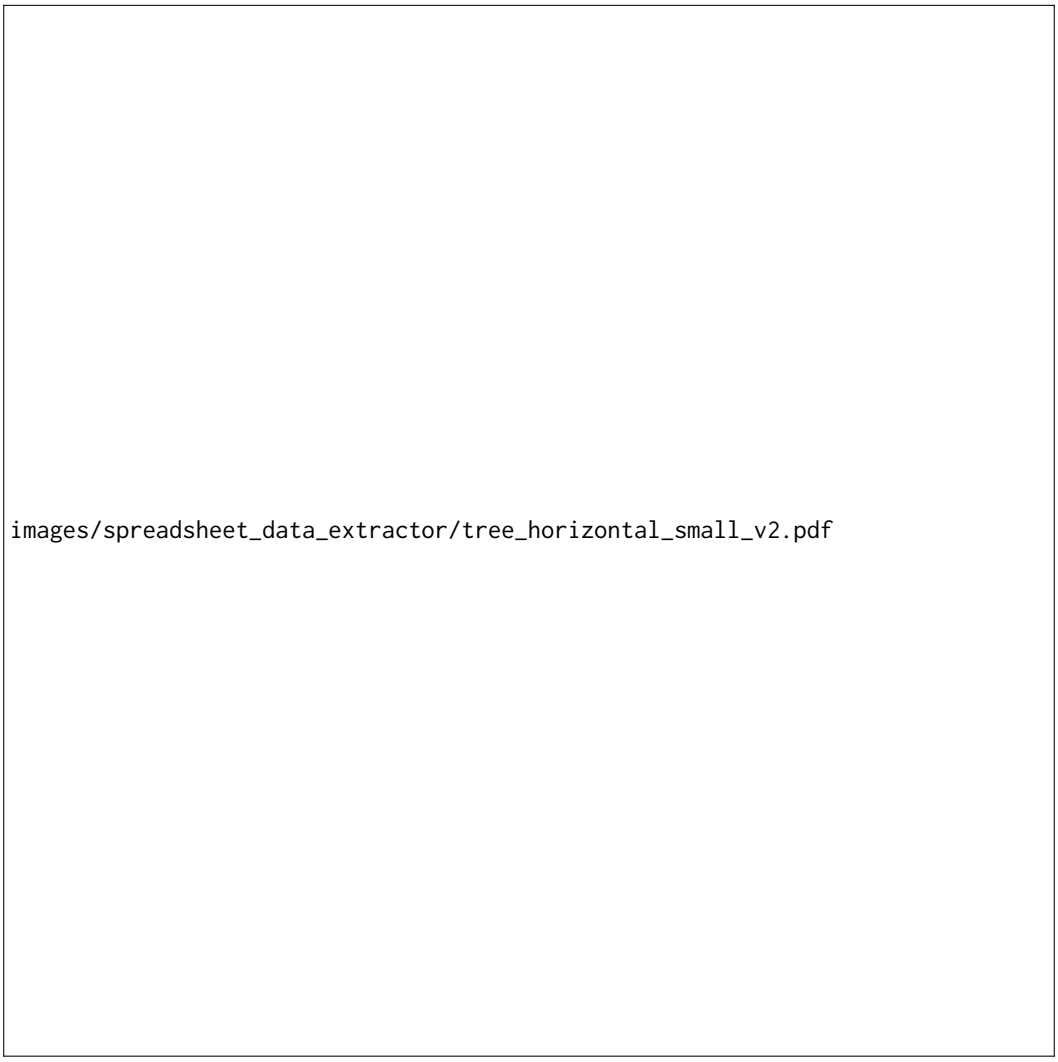
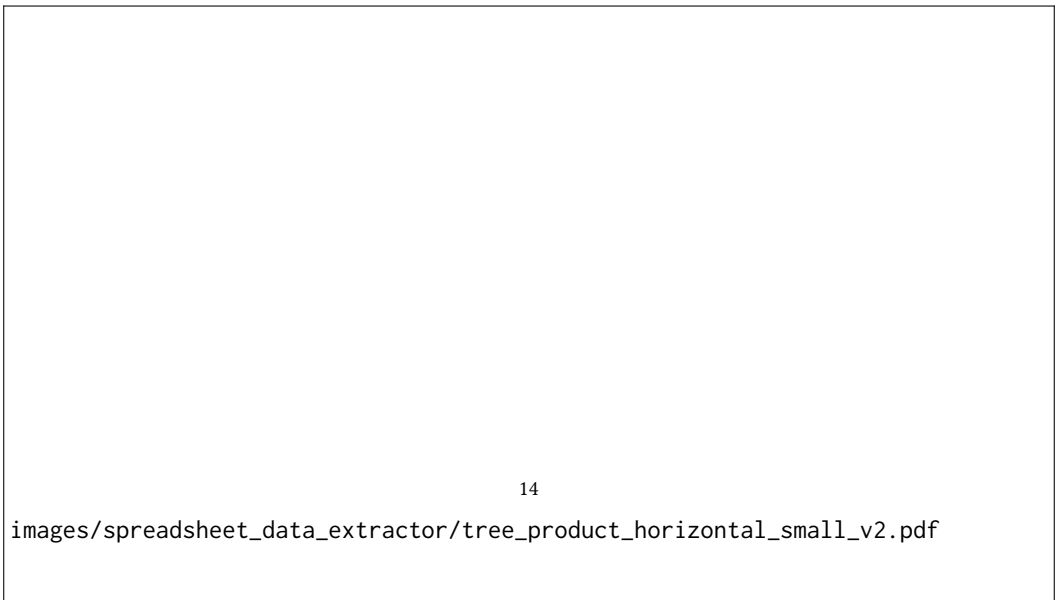


Fig. 4. The data selection view, once all selections are completed

Fig. 5. Selecting the tables arranged one below the other



(a) Tree structure of cell sets of subtree Germany, Source: Own figure



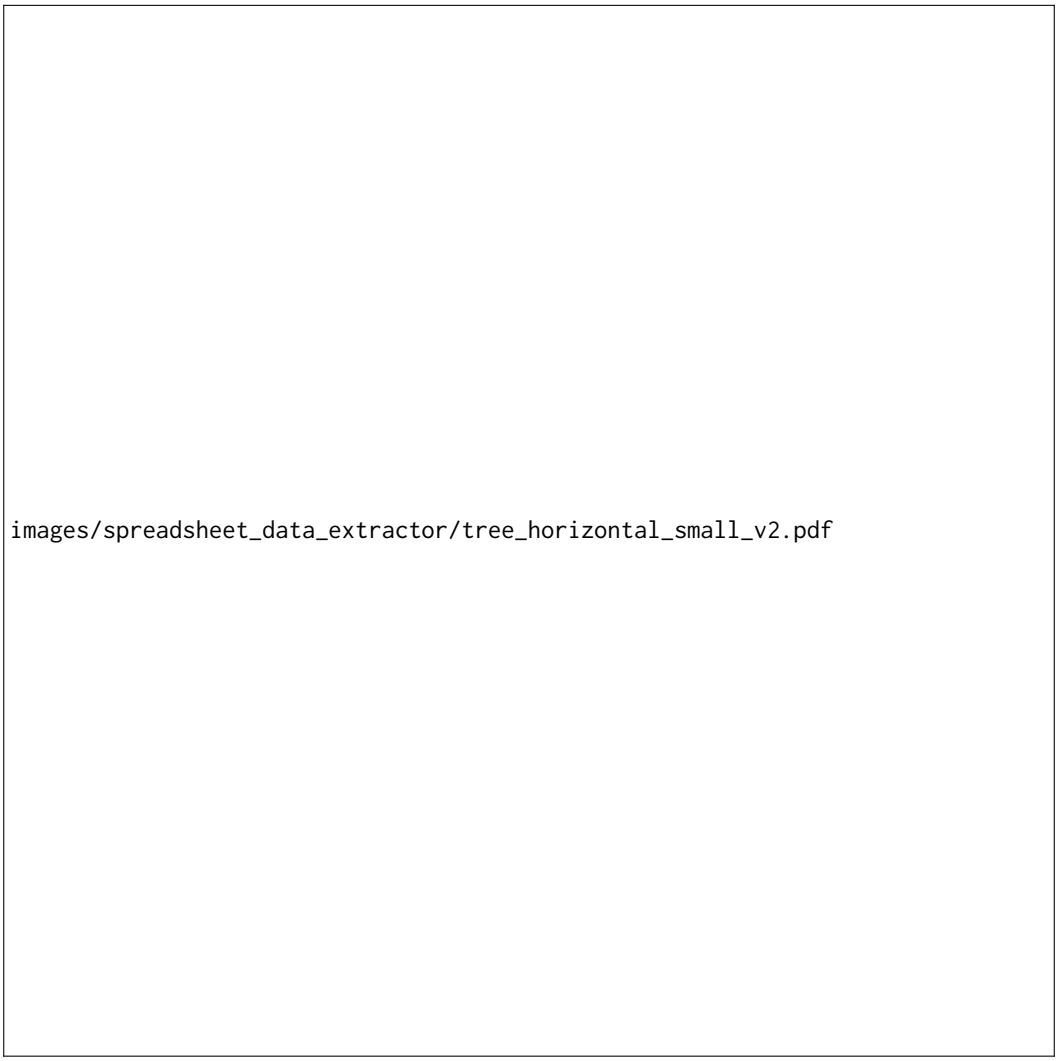


Fig. 7. Tree structure of cell sets of subtree Germany, Source: Own figure

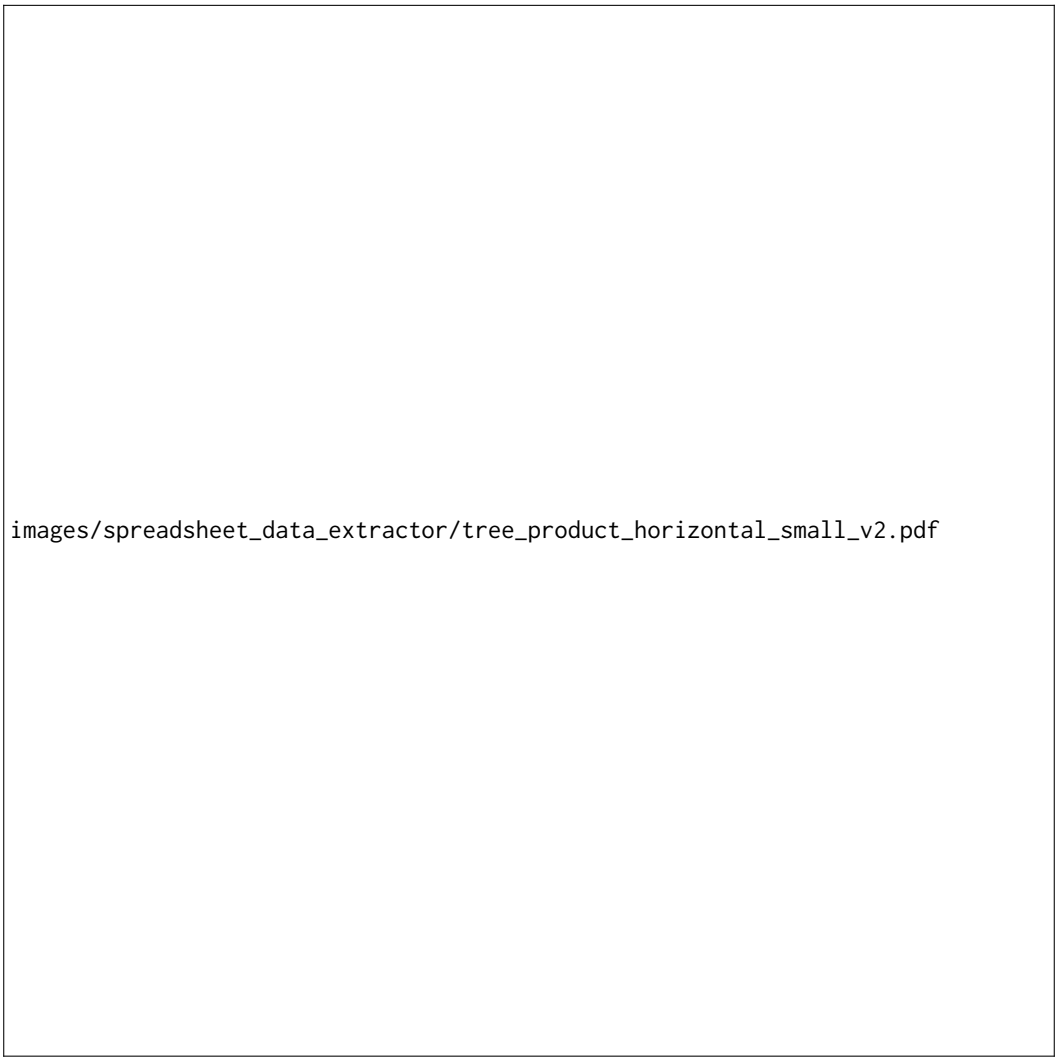


Fig. 8. Cross Product of subtree Germany, Source: Own figure

Fig. 9. Output relation, Source: Own figure

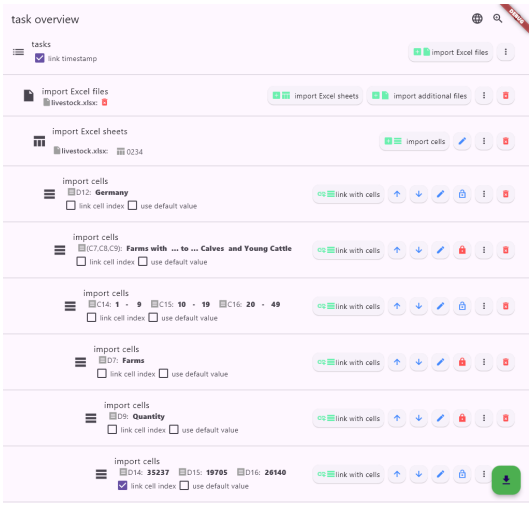


Fig. 10. The task view Listing all nodes in the hiarachy graph, Source: Own figure

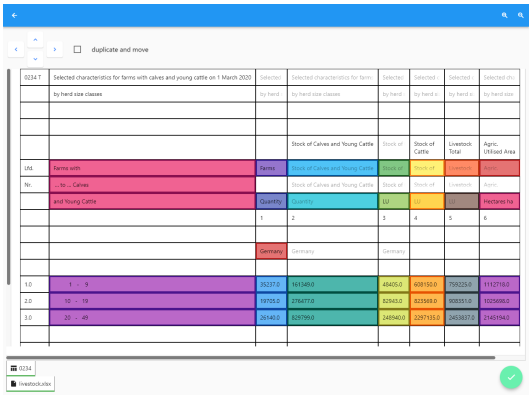


Fig. 11. The cell selection view, Source: Own figure

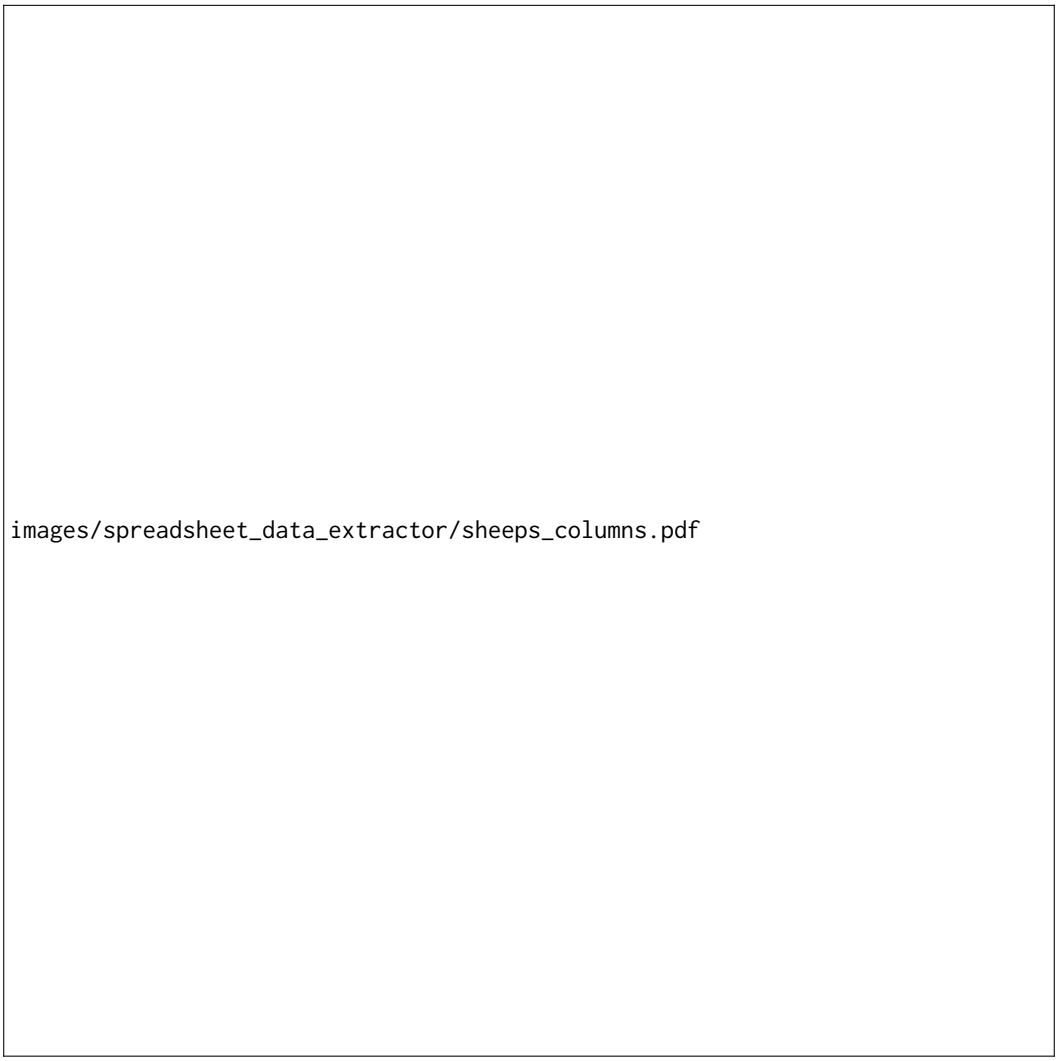


Fig. 12. Varying count of column headers, Source: Own figure

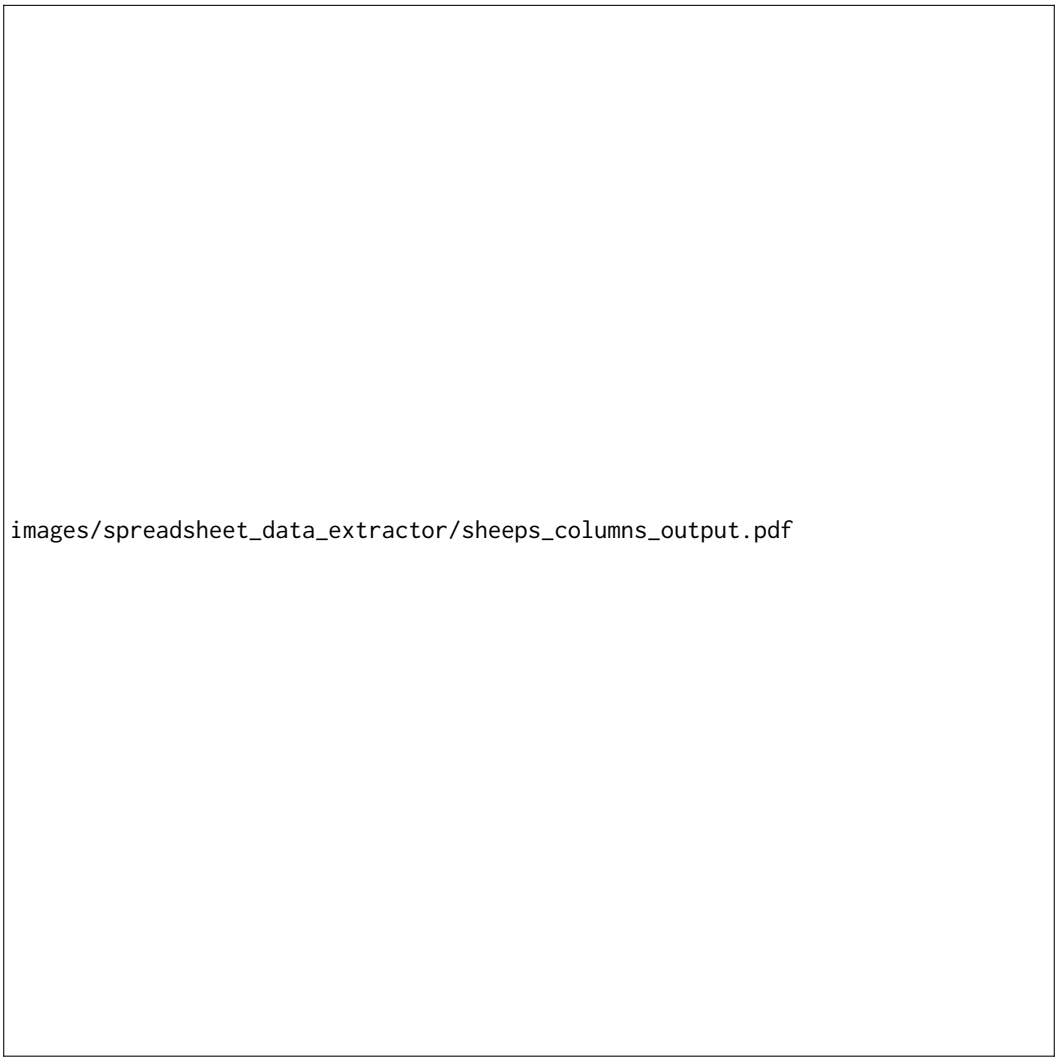


Fig. 13. Output relation with empty nodes, Source: Own figure