How to prepare a spreadsheet

Introduction

As a rule, texts that are relevant for somebody are read much more frequently than they are written. Thus the needs of the reader are way more important than the needs of the writer. The same holds for spreadsheets. Now imagine the data entered into a spreadsheet will not only be read by a human being but also by software that assists a human being. Would not everything work more efficiently and less error-prone if this software could directly use such a spreadsheet file without difficulty? We have created the following check list because in our daily work we have frequently encountered spreadsheets that were unnecessarily difficult to interpret by a human being, let alone by a computer program. We believe that if spreadsheet files were prepared according to these few and simple rules these files could way more safely and efficiently be processed.

Be aware that we here only cover a general notion of well-shaped spreadsheets. For tables to be used for statistical analysis, additionally consult https://github.com/Quartz/bad-data-guide or similar sources. The article available at http://dx.doi.org/10.18637/jss.v059.i10 provides details on transforming “messy data” into “tidy data” that can statistically be analysed. However, compared to the comfortable situation where data are only “messy” we here deal with data that are “insane”, i.e. data sets that render even seemingly simple tasks impossible such as selecting rows, mapping names, or merging data sets using common identifiers. “Messy” data sets as analysed by Wickham (2014) do have a defined structure, just not the right one, whereas the structure of insane data sets is not even obvious.

Check list

1. Indicate all relevant information by content, not by formatting or row order

1.1. Relevant information must always be **conveyed by content, not by formatting**. The formatting (e.g., colouring) can hardly be read by computer programs. It is also tedious to keep the formatting once new data are added or present data modified. Formatting as the only means to convey information should at most be used transiently, for instance to highlight (a low number of) fields that immediately need to be revised.

1.2. Having the **relevant information encoded in the entries themselves** instead of in their formatting also allows one to filter or reorder the table. Indeed, the rows of the table might need to be reordered at any time using content in one of the columns. For instance, this is necessary to merge data sets based on a column with identifiers. Thus the order in which the data have initially been entered in the table must not play a role at all.

1.3. Ultimately, authors of spreadsheets should even **prepare for losing all manually entered formatting and ordering**. Formatting thus should best be restricted to conditional formatting done by the spreadsheet software, which would guarantee that the underlying information is also contained in some field content.

1.4. Being able to reorder any table at any time also implies that **empty rows or subheadings should not be used** to carry any meaning. Instead of using empty rows – or partially empty rows with subheadings – to indicate a grouping of rows, such a grouping should be indicated by a dedicated column. Similarly, empty columns should be avoided.

1.5. If a certain ordering is of interest that cannot be obtained by sorting according to one to several columns, then a **special sorting column** should be created containing a series of numbers that yielded this ordering. At least one column with unique values is useful anyway, as detailed below.

2. Make all tables rectangular

2.1. Each sheet must contain **only a single table** with only a single row of headers. Distinct tables must be placed in distinct sheets of the spreadsheet file. For instance, a list of explanations of the symbols used by another table must be placed in a different sheet. In other words, each sheet must be interpretable like a single, rectangular structure.

2.2. When several sheets occur, they should be given **informative names**. For instance, the first sheet might be called “data” and the second one “explanation”. If the spreadsheet contains several data sets that are incompatible with respect to both rows and columns, they must be placed in distinct, accordingly named sheets.

2.3. Within a single sheet, every entry must have a **column header**, and vice versa. Do not create orphan fields somewhere in a column that has no header. Conversely, do not create empty columns. Empty rows should also be avoided; the reasons have been explained above.

2.4. Do **not merge fields** unless this is done consistently for entire rows or entire columns. When read by some software, a set of *N* merged fields would usually yield *N-1* empty fields, which is most likely not the intended outcome. There is absolutely no need to merge fields, as it is extremely easy with current spreadsheet software to replicate an entry over many rows or columns.

2.5. Do **not hide rows or columns**. There is no guarantee that the software that reads your spreadsheet file ignores the fields you were hiding. If rows should temporarily be ignored, indicate this is in a separate column. If rows should permanently be ignored, remove them from the file. Columns should have a specific meaning anyway, hence they should either be included or omitted but not hidden.

3. Let each column fit to its header and to itself

3.1. The content of each entry must have exactly the meaning indicated by its column header. If the entry has a different meaning, adapt the entry accordingly or use another column for that entry or rename the column. This simply corresponds to the general rule that you should **mean what you say**. Moreover, specific content will often be processed in a specific way; thus the readers of the table must be able to rely on the meaning of each column.

3.2. For the same reason, the data type of entries within **a single column must be consistent**. Do not mix, for instance, numeric and textual content, and avoid numeric columns within which some numbers use commas as decimal separator and others use dots. (The format of numeric and date values should best be consistent throughout the entire file.)

3.3. Textual columns (other than notes and comments) should use a **defined vocabulary** that is either self-explanatory or explained in another sheet or according to some electronic or literature resource that is clearly indicated. Use the same word for the same term throughout. Do not mix abbreviated and unabbreviated versions of the same term, and do not mix versions of the same term that only differ by capitalization or non-word characters. Correctly and consistently spell everything.

3.4. Despite the use of a potentially quite specialized vocabulary, **the spreadsheet file should be self-explanatory**. When the file format supports distinct sheets, all specific terms used in some column of some sheet should be explained in a separate sheet, either by describing them directly or by providing links to suitable resources. These explanatory sheets should be subject to the same formal constraints as the data sheets. (When textual, character-separated value files are used the explanations can be entered as comments, but these comments can not that easily be subjected to checking the format.)

3.5. **Avoid leading and trailing spaces**. These never have a meaning but if present might prevent a computer program from recognizing entries as identical; for instance, “abc ” is different from “abc”. Worse, depending on the input format as well as the software and its settings, leading and trailing spaces are sometimes stripped and sometimes kept. Adjacent spaces should be avoided for similar reasons.

3.6. **Consistently indicate missing information** by either empty fields or a specific indicator for missing information. In some cases it might be necessary to distinguish between fields that are just not available and cases where the meaning of the column is not applicable in principle to a certain row; only the former are typical instances of missing values.

4. Make the columns as specific as possible

4.1. For example, in a table with vertebrate (or pillow) data do not use a column called “Note” whose fields contain either “has feathers” or “lacks feathers”. Instead, use a column called “Has feathers” with only “TRUE” and “FALSE” as values. By making the columns maximally specific it is also possible to **avoid columns with overlapping meanings**.

4.2. **Do not repeat column-header information in the fields** of that column. For instance, if a column is called “D reaction”, then its fields should contain “TRUE” and “FALSE” (or “1” and “0”) rather than “D positive” and “D negative”, respectively. The latter would just be redundant and conceal the fact that this binary experimental outcome might be further processed (e.g., phylogenetically analysed) in the same way than other binary experimental outcomes.

4.3. Indeed, quite generally **redundancy must be avoided** as far as possible. For instance, avoid directly entered columns that just consist of the entries of other columns pasted together; if you need such columns, create them with formulas for dynamically adapting to new content. In many scientific applications the non-independence of data columns even violates important assumptions.

4.4. Similarly, columns with **notes and comments should be reduced to the minimum** possible number. Ultimately, authors of spreadsheets should even prepare for all notes and comments being ignored when the spreadsheet is processed. All too often an excessive amount of information in

notes or comments just indicates that little effort was devoted to appropriately code these data.

4.5. As we have seen above, avoiding columns with overlapping meaning as well as not repeating header information in the fields often yields columns that are not textual but of a boolean or numeric type. In programming, strong typing means to not mix values of distinct types. Strong typing should in some sense also be applied in spreadsheets, i.e. **a single column should not mix values that can be interpreted as distinct data types**: numbers, boolean values, or text.

5. Use identifiers and, once introduced, never modify them

5.1. There must be at least one **column with unambiguous identifiers for each row**. That is, this column must not contain empty fields, and each value must occur only once. When an already defined stable identifier (such as an identifier used by a data source) is not available, the best kind of identifier is often a numbering specifically introduced for this purpose.

5.2. **Identifiers should not carry information**. If an identifier carried some information, the views expressed by it could, at least in theory, be subject to change. To change an identifier is an atrocious idea, however, because identifiers are used to unambiguously select each record. For instance, distinct data sets might need to be merged using the contained identifiers.

5.3. In biology, **taxon names are not good identifiers** because they carry information (on hierarchical relationships) and are subject to change (due to a re-classification or the revision of an identification). If you already had used taxon names as identifiers, do not change them in the table, even if these taxa underwent a revision. (Note that a taxon name, once validly published, remains validly published. It might just not correspond to the most recent, or most favoured, taxonomic opinion any more.)

Determining the sanity of spreadsheet tables

Here we define a couple of computer functions to assess whether a table within a spreadsheet file is in accordance with some minimum requirements for regarding it as sane. More specific tests would depend on the problem domain and thus cannot generally be implemented. Modifying a spreadsheet file until it passes all tests described below is thus just a pre-washing process that can be performed by any author of any spreadsheet file. Afterwards the final cleaning must be conducted. (The German language provides the nice terms *Vorwaschgang* and *Hauptwaschgang* for these two routines.) A domain expert might be needed for fixing problems regarding the content of the tables. However, formal issues that are not covered by some spreadsheet sanity checking software but by this manual can usually fixed by any spreadsheet author.

Technical specification

The spreadsheet sanity tests should be conducted in order, and processing should stop after the first failed test. The result should comprise descriptions of all errors encountered so far, each with an indication of the column and row number when the test referred to single fields, and optionally the name, if any, of the affected column. The result should be empty when no errors where encountered.

tc00_empty_table. This asserts that neither the number of columns nor the number of rows is zero.

Empty tables are usually the result from additional sheets in the default configuration of spreadsheet software. They are not normally a serious defect, but apparently no further tests can be conducted with empty sheets.

tc01_column_headers. This checks that no column headers are missing or duplicated and that no non-missing column headers also occur as a field of their column or just as the beginning of such a field. Missing column headers are easy to fix. Like the next test this asserts that tables are rectangular.

tc02_nonemptyness. This checks that no entirely empty columns or rows are encountered. Regarding this prohibition of completely empty rows and completely empty columns see below. Like the last test this asserts that tables are rectangular.

tc03_whitespace. This tests that neither leading nor trailing nor adjacent white-space characters occur in any field. Such errors are easy to fix; for instance, in *LibreOffice* one can use the pattern "^\s+" for leading, "\s+$" for trailing and "\s{2,}" for adjacent white-space characters in regular-expression based search-and-replace to repair the fields.

tc04_missing_value_indicators. This checks that no column contains only missing-value indicators and that the characters used to indicate missing values are consistent within each column. Moreover, it is checked that the column names do not contain any missing-data indicators. It is required that missing values are indicated either by empty fields or by a set of non-word characters such as "?", "~" or "-" (but not white-space characters). Regarding an exception to this restriction (plus/minus columns) see below.

tc05_confusing_entries. This first tests that each column does not contain a mixture of values that can be interpreted as boolean, numeric or plus/minus with values that can not so be interpreted. The most frequent type is used as the standard and the deviating fields reported. This test checks for strong typing and can optionally be turned off. For the columns that pass this test and can neither be interpreted as boolean, numeric or plus/minus, it is checked that no entry differs from another entry only by non-word characters or only regarding upper case versus lower case. The same check is made for the column headers.

tc06_row_identifiers. This asserts that at least one column is present that contains unique, non-missing values throughout. If such a column was missing it would be easy to generate. For instance, in *LibreOffice* one can use "Edit" → "Fill" → "Series" to obtain a column with a series of unique numbers. A possible extensions of this test is to accept unique combinations of columns but this might cause other problems.

Discussion

There are two major issues with respect to which tables are regarded as sane by these checks that might require an explanation. We believe that the slight inconvenience these two restrictions might present to some authors of spreadsheets is by far outweighed by the greater ease they guarantee for assessing the sanity of spreadsheet files.

First, the requirement that no row and no column is entirely empty might seem overkill. However, it

is the only option to safely check for orphan fields and also a precaution against forgetting to enter values. Moreover, empty rows and columns have no apparent use in sane spreadsheets. As explained above, they should decidedly not carry any meaning. Because it is a quite frequent problem that a preamble precedes the header of some table, the spreadsheet sanity test described above allows the user to skip all rows until the first row that contains the maximum number of non-empty fields is encountered. This is not safe and not recommended either, however.

Second, the requirement to use either a certain non-word character or empty fields to indicate missing values might seem too restrictive. However, characters such as “?”, “~” or “-” (but not white-space characters) are intuitive as indicators of missing values and it is natural to assume that non-word characters alone should not be used to convey meaningful information. The only two exceptions we are aware of is (1) the tradition in microbiology to indicate positive phenotypic test results as “+” and negative ones as “-” and (2) the tradition in genetics indicate the direction of genes as on the “+” or on the “-” strand. Both “+” and “-” are also mathematical operators and an alternative, comprehensible encoding would be easy to devise. Nevertheless, using these symbols is a tough habit to break. Hence we recommend tests to not treat “+” and “-” as missing values in columns that only contain non-word characters and to interpret these as a special plus/minus data type.

Software supposed to input spreadsheet files after testing might by default use other indicators of missing values. For instance, the popular *R* environment for statistical computing (see https://www.r-project.org/; we do not think spreadsheet software itself is appropriate for statistics) uses a special *NA* object internally to represent missing entries as well as, by default, “NA” in textual input. However, the *read.table* function that comes with *R* has a *na.strings* argument, which can define any set of strings to be converted to missing entries during data input (empty strings are even by default understood as indicative of missing numeric values). The *write.table* function provides an according *na* argument. Thus fields that are empty or contain only non-word characters can easily be used to represent missing values in conjunction with *R*.

Implementation

As sample implementation is available at https://sanity.shinyapps.io/sanity/. The underlying *R* code can be obtained from the author of this pamphlet. The application supports LibreOffice/OpenOffice, Excel, comma-separated values (CSV), semicolon-separated values (SSV) and tabulator-separated values (TSV) as file formats.

Examples

In the following exemplary tables the first row and the first column (in grey) indicate the coordinates shown by some spreadsheet software and are not part of the proper table.

There are multiple problems in Table 1. Column A contains entries that only differ in case and others that only differ by non-word characters. Column B indicates missing values in two distinct way. Instead of using strong typing, it also mixes up textual and numeric content and thus the meaning of the fields (location of origin and year of origin). In column C there is redundancy, as

information from the column header is also contained in the fields. Column D should not be written as a note but split into two distinct columns with a specific meaning. Moreover “cont?” is unclear; does it mean “contaminated?”, “continue?”, “controlled?” or something else? Column E mixes textual and numeric representation of the same topic (the medium used) and thus does not employ strong typing. Column F has no header. Moreover, the meaning of “dark” and “light” overlaps with the meaning of “brown” and “yellow”, respectively. Column G is entirely empty. Column H has no header and contains only a single orphan field with important information, which should better be placed in a dedicated column.

	A	B	C	D	E	F	G	H
1	Strain	Origin	XYZ test	Note	Medium			
2	K2A	Toronto	XYZ+	grows well	98	brown		
3	BGZ	?	XYZ-	cont?	MZ+	light		delete
4	K2a	1975	XYZ+			dark		
5	BG-Z		XYZ+		101	yellow		

Table 1. An example from microbiology for a table that is insane in many respects.

Compared to the problems observed in Table 1, the version presented in Table 2 is quite sane. This does not mean there were no other issues with this table, but the problems listed for Table 1 definitively do not occur in Table 2.

	A	B	C	D	E	F	G	H	I
1	Strain	Location	Year	XYZ test	Growth	Medium	Colour	Keep	Note
2	K2A	Toronto	?	1	well	98	brown	true	
3	BG-Y	?	?	0	normal	72	yellow	false	maybe contaminated
4	K2B	?	1975	1	normal	?	brown	true	
5	BG-Z	?	?	1	normal	101	yellow	true	

Table 2. A table quite similar to Table 1 but without its multiple issues.

However, Table 1 also demonstrates that not all problems can be detected by the five spreadsheet sanity tests defined above. Indeed, spreadsheet sanity checks can only detect formal problems, but they of course cannot guarantee that the content is correct. Ultimately the author of a spreadsheet file is responsible for its content, and a spreadsheet sanity checking software does not relieve the author from her duties.

Sane spreadsheets in scientific publishing

Many data sets that accompany scientific papers and are published together with them as supplementary files are naturally formatted as tables. When it makes sense to group several tables, or tables and explanatory notes, into a single file, the LibreOffice, OpenOffice or Excel file formats are probably the best options. Such grouping also reduces the overall number of supplementary files. Otherwise CSV, SSV or TSV files should be preferred for publishing data tables. As a minimum requirement when using each of these formats as supplementary files of scientific papers

we strongly suggest that they pass the sanity checks described above. Each of the five formats is already supported by our sample implementation.

For optimal results in terms of reproducible research the spreadsheet files submitted as supplementary files should be equal to those actually used as data input for the conducted analyses. This can rather easily be established with spreadsheet files that pass the sanity checks, because then their tables can most probably be interpreted by some analysis software in a straightforward manner. When the program code conducting the analyses is also published in the supplement of a publication, readers of that publication can easily attempt to reproduce the results.

Portable Document Format (PDF) is an excellent format for publishing text and figures (standalone or embedded) in a platform-independent manner. PDF is not suitable, however, to store data tables to be explored with some software. There is no straightforward approach to free such tables from PDF files; the Tabula tool (http://tabula.technology/) comes close but even tables extracted from PDF files often require manual editing and in some cases are not readable at all. PDF thus should at most be used for data tables that are too small for being usable in some computational approach anyway.

Imprint

This manual (version from 21/06/17) has been written by Markus Göker, who has years of experience with partially or totally unusable spreadsheet files sent to him by cooperators, colleagues and students. You can contact him at sanity@goeker.org.