SDTA: An Algebra for Statistical Data Transformation

Jie Song
University of Michigan
Ann Arbor, Michigan, United States
jiesongk@umich.edu

George Alter
University of Michigan
Ann Arbor, Michigan, United States
altergc@umich.edu

H. V. Jagadish
University of Michigan
Ann Arbor, Michigan, United States
jag@umich.edu

ABSTRACT

Statistical data manipulation is a crucial component of many data science analytic pipelines, particularly as part of data ingestion. This task is generally accomplished by writing transformation scripts in languages such as SPSS, Stata, SAS, R, Python (Pandas) and etc. The disparate data models, language representations and transformation operations supported by these tools make it hard for end users to understand and document the transformations performed, and for developers to port transformation code across languages.

Tackling these challenges, we present a formal paradigm for statistical data transformation. It consists of a data model, called *Structured Data Transformation Data Model* (SDTDM), inspired by the data models of multiple statistical transformations frameworks; an algebra, *Structural Data Transformation Algebra* (SDTA), with the ability to transform not only data within SDTDM but also metadata at multiple structural levels; and an equivalent descriptive counterpart, called *Structured Data Transformation Language* (SDTL), recently adopted by the DDI Alliance that maintains international standards for metadata as part of its suite of products. Experiments with real statistical transformations on socio-economic data show that SDTL can successfully represent 86.1% and 91.6% respectively of 4,185 commands in SAS and 9,087 commands in SPSS obtained from a repository.

We illustrate with examples how SDTA/SDTL could assist with the documentation of statistical data transformation, an important aspect often neglected in metadata of datasets. We propose a system called C^{2}Metadata that automatically captures the transformation and provenance information in SDTL as a part of the metadata. Moreover, given the conversion mechanism from a source statistical language to SDTA/SDTL, we show how functional-equivalent transformation programs could be converted to other functionally equivalent programs, in the same or different language, permitting code reuse and result reproducibility, We also illustrate the possibility of using of SDTA to optimize SDTL transformations using rule-based rewrites similar to SQL optimizations.

CCS CONCEPTS

• **Information systems** → *Extraction, transformation and loading*; *Data exchange*; *Mediators and data integration.*

SSDBM 2021, July 6–7, 2021, Tampa, FL, USA

ACM ISBN 978-1-4503-8413-1/21/07…$15.00
https://doi.org/10.1145/3468791.3468811

KEYWORDS

data transformation algebra, statistical data, data documentation

ACM Reference Format:
Jie Song, George Alter, and H. V. Jagadish. 2021. SDTA: An Algebra for Statistical Data Transformation. In *33rd International Conference on Scientific and Statistical Database Management (SSDBM 2021), July 6–7, 2021, Tampa, FL, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3468791.3468811

1 INTRODUCTION

Statistical data transformation is commonly performed on raw input data to check data values, adjust outliers, re-scale selected attributes, create summaries and aggregate columns, and even to transform the structure of the data, through operations such as pivoting. It is usually the core of data preparation for analysis by data scientists, researchers, survey companies, government or even the general public. Indeed, the "T" in "ETL" is frequently statistical data transformation.

These data transformation operations are usually represented by *domain-specific transformation language*s (DSTLs) associated with transformation tools, the preference for which varies across user communities. For clinical trials data, SAS is the dominant choice; While for social science data, different sub-communities seem to make different choices. Fig. 1 shows the number of data downloads, by format, from Inter-university Consortium for Political and Social Research (ICPSR).

Each DSTL has its own data model, semantic and syntactic representations, and respective scope of transformation operations supported. These disparities raise a communication and manipulation barrier between languages. Given a complex transformation task, users may either compose a series of transformations with a chosen tool or reuse existing transformation scripts. However, it is not easy for most users even to understand transformations specified in an unfamiliar language, since each language is so different.

Example 1.1. Pivoting data is a common transformation that transposes data from multiple rows into columns of a single row, providing a better summary of data. In Figure 2, we use three alternative DSTLs, namely Stata, SQL Server and R, to create the identical two-dimensional pivot data Tu for the number of people in four age and gender groups from the source Tu.

- The embedded `reshape` function in Stata easily transforms the *long* table Tu into a *wide* one by specifying the `wide` keyword, the row variable i (*Age*), the column variable j (*Gender*) and the name of the column (*Gender*) storing values spanning the pivot table.
- There is no corresponding command in basic SQL. SQL Server introduced the PIVOT operation in 2005. Since the unique values of the column variable *Gender* cannot be captured by the pre-defined PIVOT command, *Columns* variable

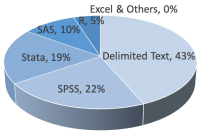


Figure 1: ICPSR data downloads by format (378,007 from Sep. 4, 2015 to Mar. 4. 2016)

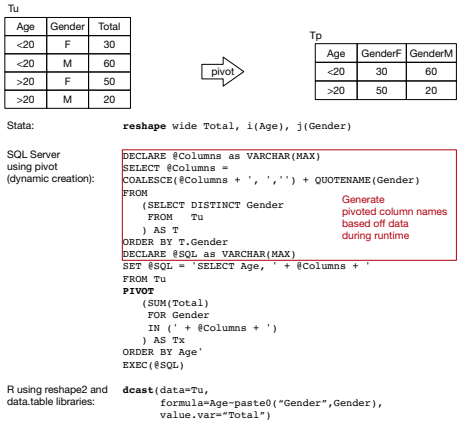


Figure 2: An example of pivot table

is first declared to dynamically select these values at runtime before pivoting. Note that the new column variable values are updated as the concatenation of *Gender* with unique *Gender* values to align with the *Tu* illustration in the figure.

- Unlike high level languages designed specifically for data transformation, R is a more powerful low level general-purpose programming language with powerful libraries for data transformation and statistical manipulation. This example leverages `dcast` function in the `reshape2` library to create a pivot table by specifying the input data table (*Tu*), the cast formula (composed of *Age* and *Gender*) and the name of the column (*Total*) storing values spanning the pivot table.

The example above illustrates that even a simple common operation can be represented in different ways shown in Figure 2 where SQL tries to compose from primitives while Stata and R offers shrink-wrapped functions. This heterogeneity creates problems not just for multi-language manipulation, but also for many basic common tasks. For example, when a user wants to understand how some dataset was derived from the original source, she may have to decipher a statistical manipulation script in an unfamiliar language. Moreover, if a large data set undergoes a long sequence of transformations there may be opportunities to improve performance by rewriting the transformation script, but it is not usually easy to figure out what rewrites make sense.

For database queries, effective optimization has been enabled by relational algebra (RA). If we could find a similar algebra for statistical transformations, we could address all the issues mentioned above.

However, developing such an abstraction is not easy. In classical relational database theory, a relation is a set of tuples, the ordering of which is not defined. Statistical data tables, while superficially similar, because of their tabular form, are inherently different from relations (and also from spreadsheets). A central characteristic of statistical data is that both rows and columns are ordered, and the ordering is essential for pattern tracking along the dimensions. This ordering prevents us from using a simple set-oriented bulk algebra like relational algebra. Due to the availability of ordering information, rows/columns can be referenced by their position in the order, which is not possible for rows in relational data. Even for columns, it is typical, in the relational world, to use name rather than position to identify a column. This is not the case in statistical data, so maintaining and manipulating the order is critical. In consequence, statistical packages attach more metadata to data elements than relational databases. Therefore a model to manipulate statistical data has to keep track of order and be more sophisticated about metadata than the relational model.

We develop a unified representation of statistical data transformations that is both *simple* and computationally efficient. The intent is to cover the vast majority of transformations, even if not all of them. Specifically, we develop an algebra, called SDTA, for manipulating statistical data transformation in a generic data model. We address two challenges: (i) how to define a generic data model that is compatible with both the relational data model and statistical data models, and allows information propagation along the transformation flow, and (ii) how to define the operators in the algebra that could preserve transformational information during language translation to facilitate metadata documentation, and at the same time benefit from optimization techniques as in relational algebra. We then develop a declarative language, called SDTL, that expresses the operators in SDTA.

Currently, both SDTA and SDTL do not support analysis operators in statistical languages.

The intellectual contributions of the paper are as follows:

- Based on the design considerations in Section 3, we build the fundamentals of statistical data transformation by defining a data model SDTDM and transformation model for statistical data transformation.
- We define SDTA as an algebraic realization of the transformation model and further illustrate SDTA operators in Section 4.
- We define SDTL as a declarative statistical transformation language in Section 5. The DDI Alliance, which maintains international standards for metadata, has adopted SDTL as one of its suite of products in 2020.
- On a collection of hundreds of diverse statistical transformation scripts, we experimentally evaluate the expressiveness of SDTL, and the fidelity of translation through an SDTL "bridge" between statistical languages in Section 6.
- In Section 7, we showcase that SDTA (and SDTL) is simple, extensible, reproducible and optimizable by use cases, including data documentation, language translation and implementation optimization. We have developed C^2Metadata (http://c2metadata.org) as a pipelined system for automatic transformation documentation as provenance-aware metadata for scientific data, available for the same four languages. Using SDTA and SDTL as bridges, we have built a mapping

between SDTL and major statistical languages (SPSS®, SAS®, Stata® and R) for efficient language translation.

Finally, we present a discussion of related work in Section 8 and conclusion in Section 9.

2 BACKGROUND

The heart of the relational data model is a well-defined small set of algebraic operations over collections of tuples. (In this paper, we will call these collections *relation*s. We will not call them "tables", since statistical tables of interest to us are also tabular.) These relational algebra operators provide the programmer with physical data independence, and facilitate efficient implementations of bulk access. Our goal is to develop a similar algebra for statistical manipulation.

Since the data is tabular in form, a first attempt may be to see if we can use relational operators, and make any minor modifications needed to these operators. However, we find several fundamental differences that prevent us from following this path.

First, row order can matter in statistical tables. Some aggregations in groups, for instance, rely on the order of tuples in group. Using Tu as an example, one may try to find the first entry whose $Total$ is greater than 30 for each *Age* group. Correspondingly for insertion, users may specify the exact relative position in which to insert a row. At first glance, matters may seem hopeless: if we are forced to use arrays, then all the physical optimizations of relational algebra will not be possible, since they rely on the (multi-)set property of relations. However, our requirement is weaker than that: we do not need to maintain an array data structure: we just have to handle relative ordering of rows when specified (which it often may not be). This ordering information, when present, needs to propagate through data transformations.

Tuples in a relation are identified by means of key attributes. No two tuples in a table can have the same value for the primary key, for example. While database management systems may permit duplicate tuples temporarily for efficiency, these duplicates are not semantically meaningful, they are not allowed by the model, and are eventually removed, In contrast, statistical tables freely permit duplicate rows. There is no concept of a key. If a tuple identifier is required, it must be supplied externally. Otherwise, tuples can be identified by relative position.

Turning to columns, we find similar issues. In relational algebra, the number of columns in a relation is fixed. Changing this requires schema update, which is a heavy-weight operation not supported as part of standard data manipulation. In contrast, columns are frequently added and deleted during statistical data manipulation. Furthermore, columns may be grouped and aggregated, just as rows are in relational algebra. Columns too may optionally be ordered. Finally, some columns may not have names or unique names: we can disambiguate between columns based on relative ordering.

We also have additional concerns not considered by relational algebra. For example, databases frequently have missing values for attributes. Relational databases use NULLs in such cases. While the consistent interpretation of NULLs is a challenge, this is dealt with at the database system level: there is no notion of a NULL in the algebra abstraction. With statistical manipulation, handling NULLs is a big concern, and is often specified in the definition of operations in statistical transformation languages of interest. In some languages such as SPSS and Stata, in addition to system-level missing values, users may define missing values to their own need for various data types, even for some selected data columns. Computations involving missing values are also handled differently by different languages. Such computations result in missing value in SPSS. In some other languages, however, missing values are ignored during computation if not specified specifically.

There are even more fundamental disparities between common statistical languages: even in the basic logic used. SAS and Stata have two-valued logic while SPSS and R have three-valued logic. When it comes to logic involving missing values, SAS treats missing values as FALSE (or negative infinity for numeric comparison) whereas Stata treats missing values as TRUE (or positive infinity for numeric comparison). SPSS and R treat them as a third logical value, neither TRUE nor FALSE. Seemingly equivalent operators in different languages are greatly affected by these disparities, often leading to unexpected results.

3 DATA MODEL AND TRANSFORMATION MODEL

Metadata can be used to document many aspects of data, including data types and data structures. Our central idea is to use metadata to capture the complexities of statistical data representation discussed in the preceding section. Towards this end, we create SDTDM that introduces adaptive level-dependent metadata. In SDTDM, metadata at different structural levels is regarded as a part of the model that transforms along with the data.

In this section, we define the data model (SDTDM) and the transformation model for generic statistical data manipulation. Data tables and transformations in common statistical languages can be converted to representation under these models minimizing information loss.

3.1 The Data Model (SDTDM)

Assume that data belongs to the domain of atomic elements containing alphanumeric strings. For metadata, we define a metadata schema similar to XML schema that shapes metadata at different structural levels, along with rules for metadata content and semantics such as what attributes a metadata element can contain, which sub elements it can contain and how many items can be present. It can also describe the type and values that can be placed into each metadata element or attribute as well as additional rules or constraints. In Figure 3, we show the schema of a Meta Table. Due to space limitation, we list a selection of common metadata names and their cardinality constraints.

Meta Table is the logical data model for statistical data transformation, a counterpart to the concept of a classical statistical table. A meta table $T = \langle meta(T), data(T) \rangle$ has table-level metadata $meta(T)$ and the table data body $data(T)$. The name of the table is a member of the table-level metadata of string type. This metadata is optional according to the illustrative data model in Figure 3. To retain the order of rows and columns, ordering functions can be defined for the two types of structural elements respectively. When converting a transformation from a source statistical language to SDTA, table-level metadata such as source file, data format, source

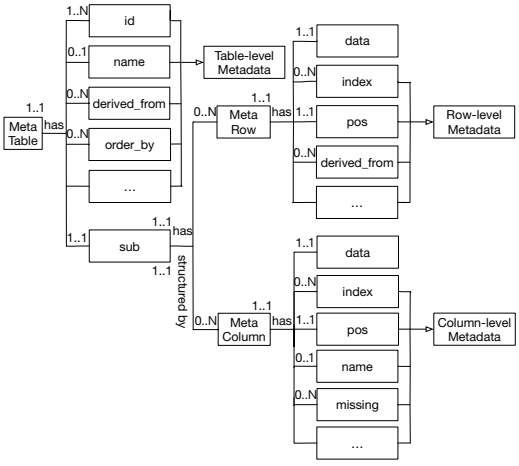
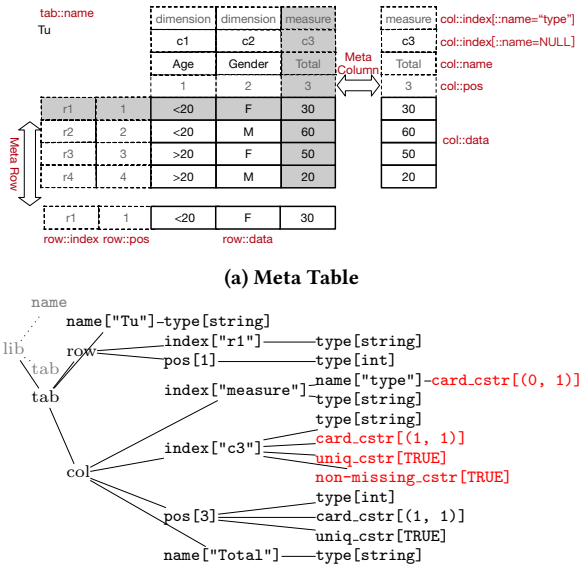


Figure 3: SDTA Data Model



(a) Meta Table

(b) Partial metadata element hierarchy

Figure 4: A Meta Table illustration of Tu

language and provenance information can be defined when needed. $data(T)$ contains Meta Rows structured by Meta Columns.

A (Meta) Column $C = \langle meta(C), data(C)\rangle$ is defined similarly with column-level metadata $meta(C)$ and the column data body $data(C)$. Since columns in statistical data are ordered, the order position pos of the column is regarded as the default identifier of a column, if no other identifiers (i.e., some required (set of) metadata with uniqueness constraint) are available. Additional column-level metadata such as multilevel indexes and missing values can be defined that propagates through transformations. They could later help with storage, searching and display of data. Meta Row is analogies to Meta Column, the definition of which is ignored here for space saving.

Definition 3.1. (Schema) Given a Meta Table T, we define

$$schema(T) = \langle schema(meta(T)), \cup_{r\in data(T)} schema(r), \cup_{c\in data(T)} schema(c)\rangle,$$

such that the schema of metadata T is composed of the schema of its metadata and the schema of its rows and columns.

Note that meta rows/columns may have differing schemas due to varying row/column-level metadata. Since the schema of meta table is determined by the schema of table-level metadata, and the union of the schemas of the rows and columns it contains, the schema of a table is table instance dependent. Operations that add, remove, or change metadata at any level may thus modify the schema. Such a design decision in the definition of the data model facilitates dynamic schema transformations that will be better elaborated in later sections.

Example 3.1. In Figure 4a, we show a partial Meta Table representation of Tu from Figure 2. In this example, pos can be regarded as the unique identifier for rows/columns.

One consequence of using a SDTDM is that when converting transformation from a source statistical language to SDTA or vice versa, maximized functional equivalent conversion is achieved during the propagation when abundant metadata is specified. In Section 7, a systematic standardization of statistical transformations for commonly used statistical languages is realized by the C^{2}Metadata system utilizing SDTDM.

3.2 The Transformation Model

To denote the transformation model, we start with the definition of *Transformation*. A Transformation specifies the algorithm to obtain a certain artefact of SDTDM, which is the result of the Transformation, starting from other existing artefacts, which are its operands. Normally the artefact produced through a Transformation is a meta table (considered at a logical level as a mathematical function).

Definition 3.2. (Transformation) Suppose that O denotes the set of operators applicable to transform meta tables. Given a set of input meta tables $\mathbf{T} = \{T_1, T_2, ...\}$ and a transformation operator $op \in O$, the derived meta table(s) $\mathbf{T_r}$ can be obtained by

$$\mathbf{T_r} \leftarrow op(\mathbf{T}).$$

op can be operators defined in SDTA or other generic operators that operates on meta tables.

Transformations defined in Definition 3.2 may introduce inconsistency in metadata unintentionally such that the metadata is not correctly describing the state of data in the derived meta table(s).

Example 3.2. The insertion of a column using SDTA operator AddCol at the ith position of a meta table (i.e., Tu in Figure 4 and $i = 3$) affects the positional order of the column originally at this position and the subsequent columns as depicted in Figure 5. Suppose that the unnamed index is a required metadata for all columns and the value of the index should be unique, the requirements are not satisfied there forth.

In our design of operators in SDTA, we deploy a similar idea that isolates data manipulation from metadata manipulation to the maximum extension in case metadata is not the operational interest

of users. We thus define *Metadata-harmonic Transformation* for consistent metadata for data in the derived meta table(s).

Definition 3.3. (Metadata-harmonic Transformation) Suppose $O^m \subset O$ is the set of operators that manipulates metadata only, V^m is the set of metadata validation rules. Given a set of input meta tables $\mathbf{T} = \{T_1, T_2, ...\}$ and a transformation operator *op*, the metadata-harmonized meta table(s) $\mathbf{T_r}$ derived from applying *op* is then

$$\mathbf{T_x} \leftarrow op(\mathbf{T})$$
$$\mathbf{T_r} \leftarrow \{op_n(...op_2(op_1(T)))| \bigwedge_{v \in V^m} \text{validate}(v, T) = \text{TRUE}, T \in \mathbf{T_x}, op_i \in O^m\}.$$

The metadata of meta tables in $\mathbf{T_r}$ should correctly describe their respective data and the process to derive the data. We name the chained metadata Transformations $op_n(...op_2(op_1(.)))$ deriving the metadata-harmonized metadata T_r as *metadata inconsistency resolution plan.*

Additional chained metadata operations could harmonize inconsistencies in metadata to ensure that the metadata for the output meta tables is accurate before propagation to the next transformation. In this case, default or customized metadata validation rules and post-Transformation metadata inconsistency resolution plans can be pre-defined for operators possible to introduce metadata inconsistencies.

Example 3.3. Suppose that type, cardinality constraint, uniqueness constraint and non-missing constraint are defined for the unnamed index of columns such that every column must have a unique string-typed unnamed index. Four possible metadata validation rules are (1) whether the value of the unnamed index is a string, (2) whether every column has one unnamed index only, (3) whether no duplicated unnamed index values exist and (4) whether the value of the unnamed index is non-missing. Continue with Example 3.2, the first two validation rules are violated after applying AddCol, op_1 adds the unnamed index to metadata (i.e., op_1 =AddMetadata) of the inserted column. As meta table $T1$ violates the fourth check rule, update its value with a unique non-missing value (i.e., op_2 = UpdateMetadata) as $T3$. The process is depicted in Figure 5.

4 SDTA

Statistical data transformations are accomplished by means of an algebra that operates on the SDTDM data model. The algebra is inspired by relational algebra and adapted to the analysis and transformation needs of general statistical operations. Next, we formally define a set of primitive operators in SDTA. Rather than simply defining counterparts within SDTA for six basic relational algebra operators (ρ, σ, π, $\times$, $\cup$, $-$), we define operators for common statistical operations. Most of the operators are composition or variants of the operators found in classical relational algebra adapted for meta table components.

Transformation on statistical data mainly manipulates three structural components: columns, rows and metadata, typically only one of these at a time. The set of basic SDTA operators operating on the SDTDM data model focuses on four aspects: the independent manipulation of (i) columns and (ii) rows, (iii) the manipulation involving the change of the table structure (both columns and rows), and (iv) metadata manipulation.The change of metadata is minimized for operators that make changes to the first three aspects, among which the orders of rows and columns are updated as identifiers.

The information flow across elements in the meta data schema permits data-metadata interaction and conversion.

4.1 Syntax

SDTA permits the declaration of variables that can range over sets of elements of the same type at the same level as depicted in nodes of Figure 3. We use a path like syntax, inspired by XPath [8], to identify and navigate elements in a Meta Table along the SDTDM hierarchy. We categorize two types of elements : (i) structural elements: meta table, meta row and meta column and (ii) descriptive elements: metadata elements at different levels.

Definition 4.1. (Element Selection Notations) An element is selected by following a path. Suppose the set of element types in $schema(T)$ is E, the set of structural element types $E^s = \{tab, row, col\} \subset E$ and the set of descriptive element types $E^m = E \setminus E^s$.

(1) Structural element selection, denoted by $x \rightarrow y$, where $x, y \in E^s \cup \{\perp\}$, selects structural elements y that are children of structural element x. When $x = \perp$, it selects the root structural element y. When $y = \perp$, it selects all children structural elements of x.
(2) Descriptive element selection, denoted by $x :: y$, $x \in E^s$, $y \in E^m \cup \{\perp\}$, selects descriptive elements y that are children of structural element x. When $y = \perp$, it selects all descriptive elements that are children of x.
(3) Predicates, denoted by $[p]$, are used to find a specific element or an element that contains a specific value. They are always embedded in square brackets.

Note that SDTDM can be further extended for “higher-order" and “lower-order" elements depending on the data structural need and description need. In SAS, for instance, there is a definition of data library, which specifies a collection of data, similar to relations in RDBMS. We can then define “higher-order" structural elements, i.e. meta lib $L = \langle meta(L), data(L)\rangle$ for a collection of meta tables. Lower-order extensions aims to help with completion of metadata descriptions. The metadata hierarchy in Figure 4b, for instance, defines meta-metadata `value`, `type`, `card_cstr` and `uniq_cstr` for metadata `pos` of column $Total$. Even more lower level descriptive elements (i.e., `value` and `card_cstr` for index name of column $Total$) could develop along the hierarchy if necessary.

Example 4.1. Given a partial metadata hierarchy example of meta table Tu in Figure 4b, we show some path expressions for its elements selection.

(1) Select the meta table named “Tu":
$$tab[:: \texttt{name} = \text{“}Tu\text{”}]$$
(2) Select the name of meta columns that have integer typed structural element children of the meta table named “Tu":
$$tab[:: \texttt{name} = \text{“}Tu\text{”}] \rightarrow col[:::: \texttt{type} = int] :: \texttt{name}$$

The selected elements are sorted by default based on the sorted order of the lowest-level structural elements they belong to.

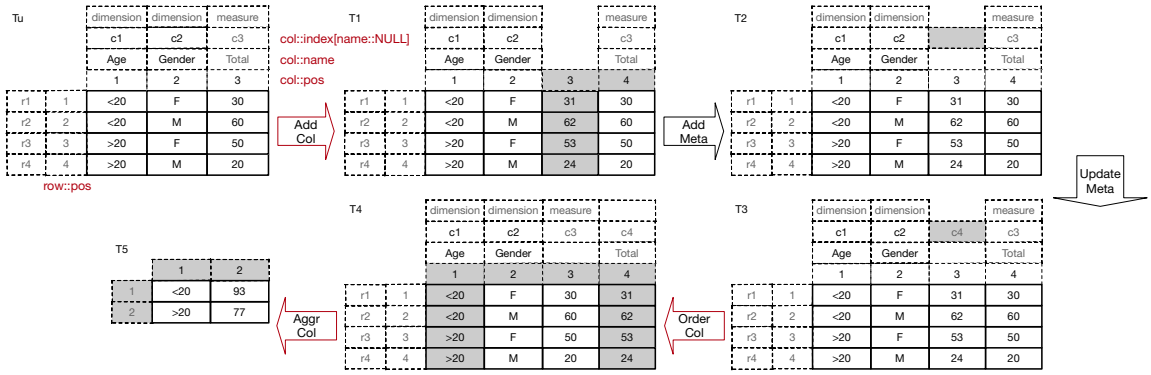


Figure 5: A meta table transformation examples

4.2 Add, Drop, Keep And Order Rows and Columns

Unlike relational theory, the columns and rows of statistical data are ordered, the preservation of ordering information after transformation is possible with the help of metadata. Since the operations on rows and columns in this subsection are alike, we define add, drop, keep and order operators for columns only. Analogous definitions for rows are hidden for space saving purpose.

AddCol and DropCol, denoted π^c and $\varkappa^c$, is a pair of operators that complements each other. The former add a data column at a particular position while the latter drops data column(s).

Definition 4.2. (AddCol) Let T be a Meta Table, v be an ordered list of values such that $|v| = MAX(row :: \texttt{pos})$ and $i \in \mathbb{N}_{MAX(row::\texttt{pos})}$. AddCol adds column data v at position i to T as

$$T_r = \pi^c_{v,i}(T).$$

v could be an ordered list of constants, or derived from applying statistical functions to elements of T. This operator increments $col :: \texttt{pos}$ by 1 for all columns whose $col :: \texttt{pos} >= i$, and then adds $col :: \texttt{pos} = i$ to the inserted column.

Example 4.2. To derive $T1$ in Figure 5, add up $Total$ value with row order for each row as the values of the new column inserted as the third column of $T1$ such that

$$T1 = \pi^c_{col[::name=\text{“}Total\text{”}]::data+row::pos,3}(Tu).$$

The column data in this example is derived from both data and metadata.

Definition 4.3. (DropCol) Let T be a Meta Table, c_x be a unique identifier of a column of T. DropCol drops column(s) $c_1, c_2, \ldots$ from T as

$$T_r = \varkappa^c_{(c_1,c_2,\ldots)}(T).$$

This operator decrements $col :: \texttt{pos}$ of the remaining columns by number of columns in $(c_1, c_2, \ldots)$ whose $col :: \texttt{pos} < i$.

Note that AddCol and DropCol cannot be imitated with canonical projection which requires the knowledge of the schema of the columns in the input and/or output meta table, i.e., the column names. As for AddCol and DropCol, only column identifier of the columns directly involved is necessary. Other columns are copied from the input meta table.

KeepCol is analogous to Selection (σ) in RA, but defined for columns.

Definition 4.4. (KeepCol) Let T be a Meta Table, p be a well-formed Boolean keep condition. KeepCol keeps column(s) of T that satisfy condition p as

$$T_r = \sigma^c_p(T).$$

Statistical languages, in particular, have heterogeneous default ordering comparison rules even for the same type of data as mentioned previously. With order being an inherent concept of SDTDM and a common need, we make frequent use of the OrderCol operator. Some RDBMSs readily provide an ordering or numbering operators[25]. Some have an inherent numbering feature as they operates on ordered relations [6].

Definition 4.5. (OrderCol) Let T be a Meta Table, o_x be an ordering pair $\langle v_x, comp_x \rangle$, where v_x is a list of values such that $|v_x| = MAX(col :: \texttt{pos})$ and $comp_x$ is an comparison function for ordering v_x. OrderCol reorders columns of T by $o_1, o_2, \ldots$ as

$$T_r = \omega^c_{(o_1,o_2,\ldots)}(T).$$

The ordering function can be a list of elements from column-level schema with their respective comparison function. This operation updates $col :: \texttt{pos}$ by the newly sorted order instead of physically moving the rows. Other operators may behave in an order-preserving way if the argument taken is ordered. Otherwise, it behaves in an order-cavalier way. We do not extensively discuss the difference between the two here. For the order-preserving version of the operators, the derived meta tables follow a natural order defined for the input meta table(s).

Example 4.3. In Figure 5, columns of $T3$ are ordered by column index named “type" and the stringtified row data whose pos is 1 such that $o = (\langle col :: \texttt{index}[:: \texttt{name} = \text{“}type\text{”}], \rangle, \langle str(row[:: \texttt{pos} = 1] :: \texttt{data}), \rangle)$. For strings, the default comparison function performs a case-sensitive ordinal comparison. Alternatively, pre-defined customized comparison functions can be used for varying ordering needs.

4.3 Aggregate Rows and Columns

In statistical analysis, other than the original data, summary data information is a primary feature achievable by statistical aggregation functions such as MAX, AVG, MEAN. Extending GROUP BY in

RA, in addition to the vertical aggregation (column aggregation) in relational algebra, horizontal aggregation (row aggregation) is also a common practice. Here we define aggregation with an optional grouping clause for columns, omitting similar definition for rows. We further define an extensible function library for complex statistical computation need of statistical data in addition to the simple aggregation functions mentioned above. The details are elaborated in the full documentation of SDTA.

Definition 4.6. (AggrCol) Let T be a Meta Table, c_x be a column specified by a unique identifier and f_x be a column-wise aggregation function, AggrCol aggregates $f_1, f_2, \ldots$ over groups specified by unique value combinations of $c_1, c_2, \ldots$ as

$$T_r = {}_{(c_1,c_2,\ldots)}\,\gamma^{c}_{(f_1,f_2,\ldots)}(T).$$

This operator updates col :: pos of T_r columns following order $(c_1, c_2, ..., f_1, f_2, ...)$. The order of rows are sorted by the values of $(c_1, c_2, ...)$.

Example 4.4. For $T4$ in Figure 5, the Transformation below computes the in-group sum of row order and value of the column named "Total" for groups categorized by values of the column named "Age".

$$T_r = {}_{(col[\text{::name=}\text{“}Age\text{”}])}\,\gamma^{c}_{SUM(col[\text{::name=}\text{“}Total\text{”}])}.$$

4.4 Join and Reshape

We further define Join and Reshape operators that manipulates data only. The order of columns and/or rows in the result are often not in the desired order. They can thus be combined with subsequent OrderRow, OrderCol operators.

The join operation is similar to the join in relational algebra.

Definition 4.7. (Join) Let T and S be Meta Tables, k_x as a column specified by a unique identifier, regarded as the join key (by default the list of join keys are the column intersection of T and S). Consider each pair of rows r_t from T and r_s from S by looping over rows in T and then in S in order, if r_t and r_s have the same value on each of the columns specified by C, add a row r to the output Meta Table T_r, where r has the same value as r_t on T and r has the same value as r_s on S. The Join operation can be represented as

$$T_r = S \bowtie_{(k_1,k_2,\ldots)} T.$$

Apart from column- or row- wise operators, the reshape operator interacts with both rows and columns by converting rows into columns and vice versa. When called with aggregation functions, it provides a close comparison of specific columns or rows for statistical inspection. In other cases, it prepares data for downstream analysis that requires specific input data format, such as repeated measures data using approaches like repeated measures ANOVA/GLM (the multilevel model) or the linear mixed model. This class of operation has no unified reference, commonly known as pivot/unpivot, transpose, wide to long/long to wide in different DSTLs, each supports similar functionalities with different supporting power. Creating two-way (like SQL Server PIVOT operator) or multi-way aggregation table (like the more powerful long to wide operator) and its reverse operation can be achieved through this operator. Here we show ReshapeToCol, the reverse operation ReshapeToRow is suppressed due to space limitation.

Definition 4.8. (ReshapeToCol) Let T be a Meta Table, c_x be the column kept, idx_x be the index column whose values are category labels and v_x be column whose values associated with category labels to be reshaped into new columns. ReshapeToCol restructures values of $v_1, v_2, \ldots$ into output Meta Table T_r whose new columns are categorized by unique combinations of values of $idx_1, idx_2, \ldots$ as

$$T_r = {}_{(c_1,c_2,\ldots),(idx_1,idx_2,\ldots)}\,\lambda^{c}_{(v_1,v_2,\ldots)}(T).$$

This operator adds a col :: index named by col :: name for each idx_x column of T to T_r. col :: pos and row :: pos are sorted such that rows and columns are ordered by tuples consists of value permutations of $(c_1, c_2, ...)$ and $(idx_1, idx_2, ...)$, respectively.

The operator encompasses a dynamic conversion from data to metadata.

Example 4.5. We represent the pivot transformation in Figure 2 as

$$Tp = \rho_{col[\text{::name,}\text{“}Gender\text{”}]+col[\text{::index[::name=}\text{“}Gender\text{”}]}(\\ ([\text{::name=}\text{“}Age\text{”}]),([\text{::name=}\text{“}Gender\text{”}])\lambda^{c}_{([\text{::name=}\text{“}Total\text{”}])}(Tu)).$$

ρ updates the names of columns by values of the index named "Gender" prefixed by "Gender".

4.5 Manipulate Metadata

As mentioned before, data manipulation is designed to be isolated from metadata manipulation to the greatest extent. The derived meta table(s) cannot inherit metadata of input meta table(s) directly as they do not accurately describe the derived data. By defining metadata check rules and metadata inconsistency resolution plans, which can often be predefined for effort saving, metadata of the derived meta table(s) can be corrected by metadata-harmonic transformation. To manipulate the metadata of Meta Tables, we define three operators: CreateMeta, DeleteMeta and UpdateMeta.

Definition 4.9. (CreateMeta) Let T be a Meta Table, CreateMeta adds metadata element e to T as

$$T_r = \alpha_e(T).$$

Definition 4.10. (DeleteMeta) Let T be a Meta Table, DeleteMeta deletes metadata e from T as

$$T_r = \wp_e(T).$$

Definition 4.11. (UpdateMeta) Let T be a Meta Table. UpdateMeta updates metadata e of T by e' as

$$T_r = \rho_{e,e'}(T).$$

Example 4.6. To resolve the metadata inconsistencies in $T1$ introduced from AddCol (shown in Figure 5), we apply AddMeta and UpdateMeta to add and assign "c4" as the value of the unnamed index of the third column. The cardinality, uniqueness and non-missing value constraints of the column are then satisfied.

$$T2 = \alpha_{col[\text{::pos=3]::index[::type=}string\&\text{::card\_cstr=(1,1)}\&\\ \text{::uniq\_cstr=}TRUE\&\text{::non-missing\_cstr=}TRUE]}(T1)$$
$$T3 = \rho_{col[\text{::pos=3]::index,}\text{“}c4\text{”}}(T2)$$

The metadata operators above are defined for meta tables. For

Table 1: Major transform commands supported by SDTL

dataset level	load, save, rename, create, merge, match, transpose, format display
column/variable level	recode, rename, aggregate, compute, delete, label, sort, missing value, join
row/case level	select, sort, add, delete, aggregate
cell level	update, label
procedural	if-then, do repeat loop

"higher-order" metadata concepts of meta table, meta library/database, set of meta libraries/databases, etc., and associated notations, the same set of metadata manipulation operators can be generalized.

5 SDTL

We further propose a declarative language SDTL for statistical data transformation based on the algebraic operators defined for SDTA.

SDTL is defined by the Convention-based Ontology Generation System (COGS) [10] information model in JSON format. It supports the most basic and widely used data transformations in four main statistical languages from major data production projects including the General Social Survey , the American National Election Study ICPSR and DataOne . Each operation is bundled with a natural language interpretation template for better human understanding.

The majority of SDTL commands are composite operations derived from nested SDTA operations, providing shortcuts for commonly used transformations of statistical data, permitting code reuse and sharing with ease. There are five major categories of operators manipulating: (i) input/output, (ii) data structure, (iii) data contents, (iv) change of metadata and (v) procedural control. We list a few commands for each category in Table 1. New operators based on SDTL and/or SDTA operators easily can be extended. SDTL also defines statistical computing functions in multiple categories available in an extensible *function library*. It supports arithmetic operations, logical conditions, aggregation and collapse operations, etc. The SDTL function library maps functions in other languages into their SDTL equivalents. Since all functions follow the same basic syntax, applications that parse other languages can translate functions into SDTL with a minimum of programming code.

An important Expression in SDTL is FunctionCallExpression calling functions defined in the function library. Current function library defines string operators, Boolean operators, validation operators, conditional operators, statistical operators, etc. Statistical functions span a wide variety of commonly adopted in statistical analysis used in major statistical languages. Some of them work on individual data cells, such as missing_value that convert the cell value into missing value. Aggregation functions are generalized to make horizontal and block aggregation possible, in addition to the traditional vertical aggregation in SQL. An example of the horizontal aggregation function working on rows is row_first, finding the first non-missing value among values of a range of columns for each row. Another set of popular statistical functions is the collapse function creating summary statistics of block data. col_sd is a simple collapse function that computes the standard deviation of multiple columns within group. Order-dependent functions are also carefully defined. A function that computes the difference between the current row and its precious row of an attribute is order-dependent. Other functions such as running sum and running average are cumulative aggregates that are both order-dependent and size-preserving. On the other hand, a function that takes the first n values of an attribute is order-dependent but not size-preserving. Order-dependent column-wise functions are also defined for rows.

The DDI Alliance, which maintains international standards for metadata, has adopted SDTL as one of its suite of products. DDI metadata is widely used by data repositories serving social sciences and natural sciences for data discovery tools, catalogs, and codebooks. SDTL provenance can be inserted into existing data derivation fields in the DDI metadata standards. The DDI Alliance will assure that SDTL is maintained and expanded in an orderly way. More details are available at the project website (http://c2metadata.gitlab.io/sdtl-docs/master/) and in [1].

6 EXPERIMENTAL EVALUATION

Given a statistical transformation script, we would like it to be represented in SDTA in its entirety. We seek to evaluate experimentally the *expressiveness* of the SDTA operators. Can they indeed capture statistical transformations used in practice? Second, we would like to know that there was no loss of information in this representation. Can we recover the original script from the algebraic representation? Or equivalently, can we correctly derive an equivalent script in a different language? We call this *fidelity*.

In this section, we evaluate the expressiveness and fidelity of SDTL for real scripts in SAS and SPSS, two of the more popular languages.

6.1 Data

We focused on the social science domain, because it is rich in published transformation scripts, and because we have familiarity with this domain. We collected test datasets from multiple public data sources where data are published with transformation scripts in either SAS or SPSS. Some sample scripts are drawn from General Social Survey (GSS), the National Survey of Family Growth (NSFG) and the American National Election Study (ANES) used in their data preparation workflows. Other samples are used at ICPSR, DataONE, and journals like the American Economics Review that require authors to deposit program code in conjunction with replication datasets. In Figure 6, we show the descending number of commands (in log scale) by command category for 4,185 SAS commands and 9,087 SPSS commands respectively from test sample scripts. For both languages, commands involving conditions (CONDITIONAL operations for SAS and IF for SPSS) are more than any other category of commands, followed by commands involving column computations (CONDITIONAL ASSIGNMENT, ARRAY ASSIGNMENT, etc for SAS and COMPUTE for SPSS). The majority of commands are statistical computations at column and row levels, and metadata manipulations at column, row and table levels, in addition to common flow controls (by loops and conditions). SQL-style manipulations, such as aggregation and join, are rare.

6.2 Expressiveness

To examine the expressive power of SDTL, we test the accuracy of SDTL Translator translation at command level. More specifically, for all commands collected, we define translation accuracy

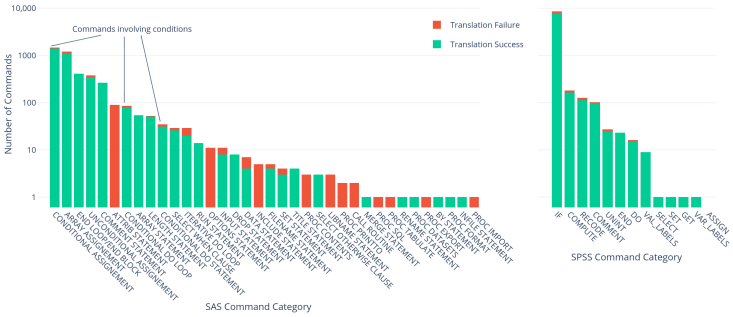


Figure 6: Command summary of test scripts in SAS (left) and SPSS (right)

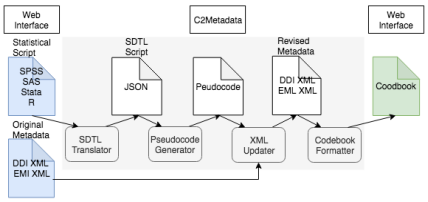
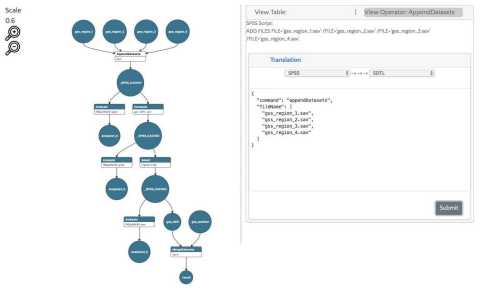
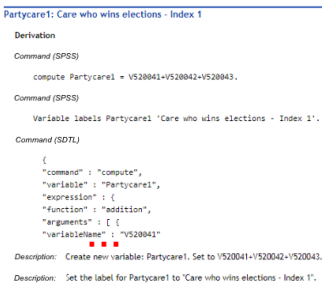


Figure 7: C^{2}Metadata workflow



(a) Graphical visualization at dataset level



(b) Codebook setting at variable level

Figure 8: Transformation Tracking Examples

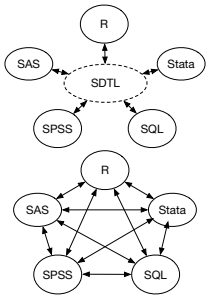


Figure 9: Bridged translation vs. pairwise translation

for command category c as

$$ACC(c) = \frac{|\text{Commands in } c \text{ correctly translated to SDTL}|}{|\text{Commands in } c|}.$$

Correct translation means that the Lexer reads the command syntax code without error, the Parser creates abstract syntax tree (AST) without error, and the AST walker outputs SDTL consistent with the original command. When an error occurs, the command in the source language is kept with corresponding error message. Though system may continue operating to completion, we consider such translations as failures. We test the accuracy of translation for SAS and SPSS respectively.

The stacked bar charts in Figure 6 depict commands correctly translated in green and wrongly translated in red. Some of the all-red command categories, ATTRIB STATEMENT in SAS for instance, are those whose translations are not yet supported in the current version of the corresponding language translator. SDTL translators for SAS and SPSS commands have an overall accuracy of 86.1% and 91.6% including unsupported categories, and 88.6% and 91.6% excluding unsupported categories. Command categories involving statistical computations have a relatively lower accuracy (less than 88%) for both languages due to limited scope of function library. Current implementation of function library supports commonly used statistical computation only. To improve accuracy, long tail operations can be easily added to the function library. Another factor affecting accuracy is the limited support for command parameters. Not all parameters have their equivalents in our SDTL implementation, even if they are theoretically expressible in SDTA.

6.3 Fidelity

We test the validity of using SDTL as the bridge for statistical language translation from the source language to the target language as mentioned in Section 7.2. To eliminate the effect of erroneous translation in the first translation phase that converts commands in the source language to SDTL, we define the fidelity as

$$FID = \frac{|\text{SDTL commands correctly translated to target}|}{|\text{SDTL commands correctly translated from source}|}.$$

The fidelity is tested for two cases: translation from (1) SAS and (2) SPSS to SDTL and then back to SPSS.

For the first case, we have no access to the ground truth of functionally equivalent SPSS to all SAS commands collected. We evaluate whether the same transformed data can be reproduced from input data using the command in the source language and the translated command in the target language. If it is a hit, these commands are functionally equivalent. As not all commands tested for expressiveness have associated data available, we selected 134 SAS commands among commands with data, all of which have been correctly translated to SDTL. During the translation from SDTL to SPSS, a fidelity score of 85.1% is achieved. Part of the problem is caused by unrecognized parameters in SDTL.

In the second test case, we compare whether the converted command is the same as the input command for identical source and target languages. Among all SPSS commands correctly translated to SDTL, almost all can be converted back to their functionally equivalent commands in SPSS, if not to the original commands. This high fidelity is ensured by reversing the conversion mechanism from the source language to SDTL.

7 USE CASES

In the preceding section, we showed that SDTA does a very good job of representing statistical transformation programs seen in practice. However, this representation has value only if it is beneficial in some way. In this section we present several application scenarios where this is indeed the case. We have already built substantial systems for the data provenance and language translation applications discussed next. After that, we briefly discuss three other uses: execution optimization, data integration and dataset search.

7.1 Transformation Documentation

As the research community responds to increasing demands for public access to scientific data, the need for improvement in data documentation has become critical. Accurate and complete metadata is essential for data sharing and interoperability [13]. However, the process of describing and documenting such data has remained a tedious, manual process even when data collection is automated.

Researchers in many fields use statistics packages for data management as well as analysis. These packages, however, lack tools for documenting variable transformations in the manner of a workflow system or even a database. At best, the operations performed by the statistical package are described in a script, which more often than not is not even available to future data users. Different statistics packages differ in data model, transformation representation and scope of transformations covered; complicating the understanding of the transformation process.

We have launched C^{2}Metadata [37] as a pipelined system for automatically documenting transformations as provenance-aware metadata for scientific data, available for four major statistical languages: SPSS, Stata, SAS and R. A Python version is under development. The original metadata is updated with transformation information represented in SDTL regardless of the original language used. The pipelined modular architecture of C^{2}Metadata, comprising four modules, is shown in the grey area in Figure 7, namely SDTL Translator, Pseudo-code Generator, XML Updater and Codebook Formatter. These modules convert the source language commands to SDTL representations, translate SDTL commands to natural language descriptions, update the metadata in XML formats and create a human-readable codebook for online access.

Apart from the revised documentation, C^{2}Metadata allows tracking of the processed transformations at four different levels: dataset level, variable level, case level and cell level, in multiple settings and transformation representations, along the lineage graph. Figure 8a shows the tracking of the transformation of the sample at the dataset level in a graphical setting, where each node is a dataset after one step of transformation. Here we present the transformation by the original SPSS representation. By clicking on the node, the intermediate transformation result will expand for inspection. Variable-, case-level and cell-level give more provenance information of the object of interest at the current stage or along the transformation trace. An example of the tracking of variable Partycare1 is shown in Figure 8b in a codebook setting. In this scenario, all transformations applied to this variable are represented by the original language (SPSS), SDTL and a human-readable natural language description.

We are extending C^{2}Metadata by collaborating with DataOne's ProvOne project and the NCSA BrownDog project to promote tracking data transformations in the broader scientific data community.

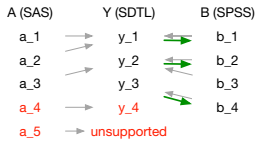


Figure 10: Translation illustration from SAS to SPSS via SDTL

7.2 Language Translation

In language translation, a *pivot language* (or *bridge language*), can be used as an intermediary language for translation between many different language pairs. Using SDTL (or SDTA) as the bridge, we could build a bidirectional translation mapping between the pivot language and major statistical languages, instead of numerous pairwise mappings, as shown in Figure 9 . Such translation could be valuable for user comprehension, code reproducibility, and execution efficiency (when some operations are more efficient to implement in one language). For each transformation, we expect that the output data should be equivalent no matter which language is used to represent the transformation. Here we mean equivalence in ordered data ignoring metadata.

Even if one operation in the source language can be converted to many functional equivalent representations in the pivot language, we construct a many-to-one mapping to simply permit the translation. We then build a one-to-one mapping between operators in the pivot language to those in the target language ignoring heterogeneous conversion possibilities. The translation process from SAS to SPSS can be illustrated by Figure 10. With the help of SDTL Translator in C^{2}Metadata that converts source languages to SDTL via abstract syntax tree (AST) templated mappings, we have the initial many-to-one mappings (grey arrows) from SAS/SPSS (A/B) to SDTL (Y). To convert from SDTL to SPSS, for each y_i with at least one mapping from a set of $\{b_j\}$ in B, we choose one of them as the deterministic reverse mapping from y_i to b_j (green arrows). a_4 and a_5 fail the translation as no mapping from operators in SPSS to y_4, and a_5 maps to Unsupported in SDTL. The translation accuracy is thus dependent on the expressiveness of SDTL and the fidelity of initial mappings from SAS/SPSS to SDTL, which are evaluated in Section 6. In Figure 11, we show an example of the translation.

Currently, we mind these minor disparities across languages in terms of data types, missing values, logical expressions, and functions defined in function libraries. Since we could not name all translation mappings, to ensure equivalent output data, a post-translation validation step executes the original operation in the source language and the translated operation in the target language for comparison. Mappings of cases failing the validation are reconsidered during the development workflow. Some source language operations may not be fully expressed in the target language. For example, one can add label to rows in SAS but not in SPSS. These complications are generally caused by the undefined base operation or undefined unit of operation (SPSS has no definition of row label in its data model) in the bridge language or in the target language.

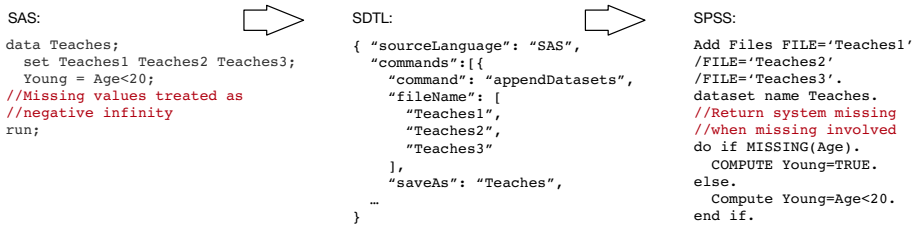


Figure 11: Sample of translation deriving *Teaches* from SAS to SPSS via SDTL

7.3 Optimization, Integration and Search

Statistical data transformation represented by SDTA operators can facilitate optimization of execution . We consider two cases: execution optimization for (1) a single transformation and (2) a series of transformations. As statistical transformations are generally written in scripts, bulk execution of a series of transformations are performed in order. These transformations take the input data or intermediate data generated from transformations one or more steps back as input. We can thus create the logical execution plan based on the lineage graph of Meta Tables, depicting what transformations need to be executed after an action has been called as constraints. The transformations can then be decomposed into primitive operators to form a query tree for optimization using classical SQL optimization techniques and order-based optimization techniques such as sort elimination [35], sort move-around [36] and other techniques for order in array databases [21].

Though data integration has been massively explored, past efforts do not emphasize enough the integration of metadata, nor the integration of transformational information. To generate more value from shared data, the emerging research around dataset search has drawn more attention on developing frameworks, methods and tools to help match a data need against datasets. Dataset search is largely keyword based over published metadata in a single repository or across multiple repositories. A standard representation of data transformations, SDTL or SDTA for instance as part of the metadata for documenting data provenance, could promote better integration and search of such datasets.

8 RELATED WORK

There is no shortage of algebras for data transformation. Both Statistical Databases (SDBs) and OLAP (On-Line Analytical Processing) are related to our work as they provide statistical summarizations over dimensions of multidimensional data, though motivated by different types of applications: socio-economic analysis and business data transactions respectively. SDBs emphasize conceptual modeling with classification structures, involving management of metadata of the category values and their hierarchical associations. They can be represented as tables, with a graph model and/or with the data cube model. SDB-specific concerns include aggregation over one or more dimensions, data sparsity, advanced statistical computations and privacy[28, 33, 34]. Both the original data and the summarized data in tabular model can be represented by SDTDM with multi-index metadata and statistical computation support.

Recent attention has been drawn to scientific data such as sensor, image and geographical data that are often *multi-dimensional* in nature (also known as raster data, gridded data or datacubes). The need of the ability to handle complex analytics based on core linear algebra operations on large data is noticed. To support arrays, array databases (such as RasDaMan[3], MonetDB[11], SciDB[38] and Google Earth Engine[14]) have been developed specifically for their storage, management and analysis. Other efforts haven been made to extend RDBMSs to support array data handling (such as PostgreSQL with PostGIS[30] and Oracle GeoRaster[26]). A number of array models and algebraic languages including AQL, AML, Array Algera and RAM [2, 22, 24, 28, 39] have been proposed and tailored for array data. Though these models provide the support for ordered data, arrays are *uniform* in that all cells in a given array have the same data structure and the operations they focus are heavily dependent on this uniform feature (such as matrix operations, sub-array extraction, reduce an array to a scalar value), which is different from the data and the operations we are targeting in this paper. Since the data stored in array databases are often large and multi-dimensional, scalability is the primary concern. There are thus many papers studying physical organization, storage-based optimization and access methods underlying specific implementations of these algebras, which we will not describe here since our focus is on the conceptual modeling of data.

More broadly, relational algebra (RA) [9] lays the theoretical foundation for SQL, which is the most widely used data manipulation language for databases. Despite its extending support for limited data transformations and analytics, it is primarily used for answering queries regarding data in situ where data can be highly normalized. Extensions of relational algebra adapt RA for various data manipulation purposes as data types, structures and other aspects of data evolve [23, 27, 31]. Works accommodating statistical analysis[28, 33, 34], ordered data[4], metadata manipulation [20, 43], tabular data structures [5], null value handling [15, 44] are more closely related to our work. Recent tools like Potter's Wheel [29], Wrangler [19], Foofah [18] provide an interactive interface for less technical users with more specific transformation needs such as data wrangling, data warehousing and data cleaning. However, their emphasis is syntactic transformation rather than preparation for statistical analysis.

More targeted tools have been widely adopted in the data science community over the past half century. During the 1980's, there were a number of studies in Statistical Databases, focusing mainly on socio-economic data in need of complex statistical operations not fully supported by relational databases. Statistical packages like SPSS® , Stata® , SAS®, R(by packages like dplyr [42], tidyverse [41] and reshape2 [40]) and Python are more popular for statistical data transformation for tabular data. Though simple data conversion (opening data file written in one language by another

language using tools like Stat/Transfer[7]) is straightforward, the transformation syntax conversion between these languages is not easily realizable. Attempts for embedding one language in other languages is not new for statistical data. Major commercial statistical software such as SAS and SPSS include a built-in interface for calling R functions[12, 17]. Other attempts [16] have been made to communicate data and results interactively in language interfacing. These attempts, however, are tailored to the characteristics of languages involved, and are often between statistical languages and a more general programming language for operation decomposition.

Standard representations of statistical data transformation have been proposed regardless of the transformation engine used. The Validation and Transformation Language (VTL) [32] is such a language that defined validation and transformation rules for statistical data at the abstract level. It primarily supports exchange of validation rules for data quality specification and validation purposes accounting for part of the data transformation purpose but not all. Since it is not information-preserving, VTL cannot serve as the bridging language for inter-conversion of statistical languages.

9 CONCLUSION

Statistical data transformations are a critical component of the data science pipeline. Due to their heterogeneity and complexity, they have been relegated to scripts operating outside the managed confines of DBMSs. Here, we introduce an algebra, SDTA, which has only a few operators covering the majority of statistical transformations. SDTA has been implemented, with syntactic sugar added, as a language SDTL used for several applications, including data documentation and translating scripts between languages. We presented experimental numbers supporting the value of our approach.

REFERENCES

[1] George Alter, Darrell Donakowski, Jack Gager, Pascal Heus, Carson Hunter, Sanda Ionescu, Jeremy Iverson, HV Jagadish, Carl Lagoze, Jared Lyle, et al. 2020. Provenance metadata for statistical data: An introduction to Structured Data Transformation Language (SDTL). *IASSIST Quarterly* 44, 4 (2020).

[2] Peter Baumann. 1994. Management of multidimensional discrete data. *The VLDB Journal* 3, 4 (Oct. 1994), 401–444.

[3] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, and N. Widmann. 1998. The Multidimensional Database System RasDaMan. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data* (Seattle, Washington, USA) *(SIGMOD '98)*. Association for Computing Machinery, New York, NY, USA, 575–577. https://doi.org/10.1145/276304.276386

[4] Peter Baumann and Sönke Holsten. 2011. A Comparative Analysis of Array Models for Databases. *FGIT-DTA/BSBT* (2011).

[5] Peter Baumann and Sönke Holsten. 2011. A comparative analysis of array models for databases. In *Database theory and application, bio-science and bio-technology*. Springer, 80–89.

[6] Peter A Boncz and Martin L Kersten. 1999. MIL primitives for querying a fragmented world. *The VLDB Journal* 8, 2 (1999), 101–119.

[7] Inc Circle Systems. 2015. Stat/Transfer (version 14). https://stattransfer.com

[8] James Clark, Steve DeRose, et al. 1999. XML path language (XPath).

[9] Edgar F Codd. 2002. A relational model of data for large shared data banks. In *Software pioneers*. Springer, 263–294.

[10] Colectica. 2017. Convention-based Ontology Generation System (COGS) 1.0. http://cogsdata.org/docs/.

[11] Roberto Cornacchia, Sándor Héman, Marcin Zukowski, Arjen P de Vries, and Peter A Boncz. 2008. Flexible and efficient IR using array databases. *VLDB J.* (2008).

[12] Catherine Dalzell. 2013. Calling R from SPSS. https://developer.ibm.com/tutorials/ba-call-r-spss/

[13] Boris Glavic and Klaus R Dittrich. 2007. Data Provenance: A Categorization of Existing Approaches.. In *BTW*, Vol. 7. 227–241.

[14] Noel Gorelick, Matt Hancher, Mike Dixon, Simon Ilyushchenko, David Thau, and Rebecca Moore. 2017. Google Earth Engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment* (2017). https://doi.org/10.1016/j.rse.2017.06.031

[15] Georg Gottlob and Roberto Zicari. 1988. Closed World Databases Opened Through Null Values. *VLDB* (1988).

[16] EF Haghish. 2019. Seamless interactive language interfacing between R and Stata. *The Stata Journal* 19, 1 (2019), 61–82.

[17] SAS Institute Inc. 2016. Calling Functions in the R Language. https://support.sas.com/rnd/app/studio/statr.pdf

[18] Zhongjun Jin, Michael R Anderson, Michael J Cafarella, and H V Jagadish. 2017. Foofah - Transforming Data By Example. *SIGMOD Conference* (2017).

[19] Sean Kandel, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. 2011. Wrangler - interactive visual specification of data transformation scripts. *CHI* (2011).

[20] Laks VS Lakshmanan, Fereidoon Sadri, and Iyer N Subramanian. 1996. SchemaSQL-a language for interoperability in relational multi-database systems. In *VLDB*, Vol. 96. Citeseer, 239–250.

[21] Alberto Lerner and Dennis E Shasha. 2003. AQuery - Query Language for Ordered Data, Optimization Techniques, and Experiments. *VLDB* (2003).

[22] Leonid Libkin, Rona Machlin, and Limsoon Wong. 1996. A Query Language for Multidimensional Arrays - Design, Implementation, and Optimization Techniques. *SIGMOD Conference* (1996).

[23] Nikos A Lorentzos and Roger G Johnson. 1988. Extending relational algebra to manipulate temporal data. *Information Systems* 13, 3 (1988), 289–296.

[24] Arunprasad P Marathe and Kenneth Salem. 1997. A Language for Manipulating Arrays. *VLDB* (1997).

[25] Jim Melton. 2003. *Advanced SQL: 1999: Understanding object-relational and other advanced features*. Morgan Kaufmann.

[26] Chuck Murray, Janet Blowney, J Xie, T Xu, and S Yuditskaya. 2003. Oracle Spatial GeoRaster, 10 g Release 1. *Oracle Corporation* (2003), 2–31.

[27] G Özsoyoğlu, ZM Özsoyoğlu, and Victor Matos. 1987. Extending relational algebra and relational calculus with set-valued attributes and aggregate functions. *ACM Transactions on Database Systems (TODS)* 12, 4 (1987), 566–592.

[28] Gultekin Özsoyoglu and Z Meral Özsoyoglu. 1985. Statistical Database Query Languages. *IEEE Trans. Software Eng.* (1985).

[29] Vijayshankar Raman and Joseph M Hellerstein. 2001. Potter's Wheel - An Interactive Data Cleaning System. *VLDB* (2001).

[30] P Ramsey, VB Columbia Refractions Research Inc, and 2005. [n.d.]. Introduction to PostGIS. *drm.cenn.ge* ([n. d.]).

[31] Mark A Roth, Herry F Korth, and Abraham Silberschatz. 1988. Extended algebra and calculus for nested relational databases. *ACM Transactions on Database Systems (TODS)* 13, 4 (1988), 389–417.

[32] SDMX. 2018. Validation and Transformation Language (VTL). https://sdmx.org/?page_id=5096

[33] Arie Shoshani. 1982. Statistical Databases - Characteristics, Problems, and some Solutions. *VLDB* (1982).

[34] Arie Shoshani. 1997. OLAP and Statistical Databases - Similarities and Differences. *PODS* (1997).

[35] David Simmen, Eugene Shekita, and Timothy Malkemus. 1996. Fundamental techniques for order optimization. In *the 1996 ACM SIGMOD international conference*. ACM Press, New York, New York, USA, 57–67.

[36] SlivinskasGiedrius, JensenChristian S, and SnodgrassRichard Thomas. 2002. Bringing order to query optimization. *ACM SIGMOD Record* (June 2002).

[37] Jie Song, George Alter, and H. V. Jagadish. 2019. C2Metadata: Automating the Capture of Data Transformations from Statistical Scripts in Data Documentation. *Proceedings of the 2019 International Conference on Management of Data* (2019).

[38] Michael Stonebraker, Paul Brown, Alex Poliakov, and Suchi Raman. 2011. The Architecture of SciDB. *SSDBM* (2011).

[39] Alex R van Ballegooij. 2004. RAM: A Multidimensional Array DBMS. In *Current Trends in Database Technology - EDBT 2004 Workshops*. Springer, Berlin, Heidelberg, Berlin, Heidelberg, 154–165.

[40] Hadley Wickham. 2007. Reshaping Data with the reshape Package. *Journal of Statistical Software* 21, 12 (2007), 1–20. http://www.jstatsoft.org/v21/i12/

[41] Hadley Wickham. 2017. *tidyverse: Easily Install and Load the 'Tidyverse'*. https://CRAN.R-project.org/package=tidyverse R package version 1.2.1.

[42] Hadley Wickham, Romain François, Lionel Henry, and Kirill MÃŒller. 2018. *dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr R package version 0.7.6.

[43] Catharine M Wyss and Edward L Robertson. 2005. Relational languages for metadata integration. *ACM Transactions on Database Systems (TODS)* 30, 2 (2005), 624–660.

[44] Carlo Zaniolo. 1984. Database Relations with Null Values. *J. Comput. Syst. Sci.* (1984).