Data Handling

Data Science Project: An Inductive Learning Approach

Prof. Dr. Filipe A. N. Verri

About these slides

These slides are companion material for the book

Data Science Project: An Inductive Learning Approach

Prof. Dr. Filipe A. N. Verri

`https://leanpub.com/dsp`

All intellectual content comes from the book and is not AI-generated. Slides were produced with the assistance of Claude Code.

† *It's dangerous to go alone! Take this.*

— *Unnamed Old Man, The Legend of Zelda*

Overview

Contents

- Formal structured data
- Data handling pipelines
- Split-invariant operations
- Other operations
- An algebra for data handling

Objectives

- Define a formal structure for tables
- Define operations on tables
- Understand split invariance

Formal structured data

Indexed table

Definition

An indexed table $T = (K, H, c)$:

- $K = \{K_1, \dots, K_k\}$ — set of index columns
- H — set of non-index columns
- c — cell function mapping (row, column) to values

- Rows indexed by tuples $r = (k_1, \dots, k_k)$
- Each row has a cardinality (repetitions of the entity)
- Values may be **missing** (?)

Nested rows and value matrices

- **Nested row**: associates different columns with the same repetition of the entity
- **Value matrix** $V(r)$: stacking nested rows into a $\text{card}(r) \times |H|$ matrix
- Value matrices are assumed minimal (no all-missing nested rows)

Split and bind

Split

Given indicator $s(r) \in \{0, 1\}$, creates two disjoint tables:

$$\text{split}(T, s) = (T_0, T_1)$$

A split **never breaks a row** — indices define indivisible entities.

Bind

Inverse of split — combines two **disjoint** tables:

$$\text{bind}(T_0, T_1) = T$$

Premises in real-world applications

- Entities represented by rows are indivisible
- Binding only for tables that do not share entities
- Data collection $\approx$ split from the universe of entities
- Fewer index columns $\rightarrow$ more information per entity required
- More index columns $\rightarrow$ restrictions on allowed operations
- Neglecting these issues leads to statistical biases

Split invariance

Definition

Operation f is **split-invariant** if for any table T and split s:

$$f(T_0 + T_1) = f(T_0) + f(T_1)$$

where $T_0, T_1 = \text{split}(T, s)$.

- Ensures the operation does not depend on how data was collected
- Avoids **data leakage** and sampling bias
- Sufficient (not necessary) condition for preventing leakage

Illustrative example — Student grades

student	subject	year	grade
Alice	Chemistry	2020	6
Alice	Math	2019	8
Alice	Physics	2019	7
Bob	Chemistry	2018	?
Bob	Chemistry	2019	7
Bob	Math	2019	9
Bob	Physics	2019	4
Bob	Physics	2020	8
Carol	Biology	2020	8
Carol	Chemistry	2020	3
Carol	Math	2020	10

Impact of index choice

Index = (student, subject):

- Cannot know if Alice took Biology
- Can confirm Bob passed Physics (second attempt)
- Cannot guarantee Carol only took classes in 2020

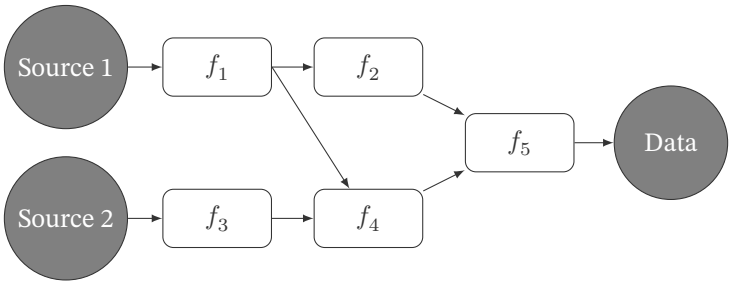
Index = (student):

- Can confirm Alice never took Biology
- No information about Bob (not in our split)
- Can confirm Carol only took classes in 2020

Fewer indices $\rightarrow$ more information about present entities.

Data handling pipelines

Data handling pipeline



Forks and merges allow complex processing.
Declarative approach enables optimization and parallel processing.

Split-invariant operations

Split-invariant operations

Operations safe to use during data integration and tidying:

- Tagged splitting and binding
- Pivoting (long-to-wide and wide-to-long)
- Joining (left join with fixed table)
- Selecting columns
- Filtering rows
- Mutating (creating new columns)
- Aggregating (reducing row cardinality)
- Ungrouping (increasing index columns)

Tagged bind and split

- **Tagged bind**: combine disjoint tables + add source column s
- Solves: single observational unit in multiple tables
- Pay attention to hidden semantics (e.g., units of measurement)
- **Tagged split**: reverse — split by values of column s
- Split-invariant by definition (each row has unique s value)

Pivoting

Long format

city	year	qty.
A	2019	1
A	2020	2
A	2021	3
B	2019	4
B	2020	5
B	2021	6

$\leftrightarrow$

Wide format

city	2019	2020	2021
A	1	2	3
B	4	5	6

- **Long-to-wide**: name column values become new columns
- **Wide-to-long**: column names become values
- Both operations are split-invariant
- Solves: headers as values, variables in rows and columns

Joining

- Combines two tables based on common index columns
- **Inner join**: only matching rows kept
  - Can “erase” rows — potential source of bias
- **Left join** (with fixed table): all rows from first table kept
  - Missing matches $\rightarrow$ missing values for second table columns
  - **Split-invariant** when second table is fixed
- Cardinality of joined rows must be constant (aggregate first if needed)

Selecting and filtering

Selecting (columns):

- Choose a subset of non-index columns; rows unchanged
- Split-invariant if predicate does not depend on values

Filtering (rows):

- Choose rows satisfying a predicate; columns unchanged
- Predicates combined with logical disjunction (or)
- Split-invariant because rows are treated independently

Mutating and aggregating

Mutating:

- Create new columns from a function of existing values
- Example: `y = x + 1`, `y = ifelse(x > 0, 1, 0)`
- Split-invariant (rows treated independently)
- Solves: multiple variables stored in one column

Aggregating:

- Ensure all rows have cardinality 1
- Function maps value matrix to a single tuple
- Split-invariant (rows treated independently)

Ungrouping

Transforms a non-index column into an index column.

city	year	qty.
A	(2019, 2020, 2020, 2021)	(1, 2, 3, 4)
B	(2019, 2020, 2021)	(5, 6, 7)

↓ ungroup by *year* ↓

city	year	qty.
A	2019	1
A	2020	(2, 3)
A	2021	4
B	2019	5
B	2020	6
B	2021	7

Other operations

Projection (grouping)

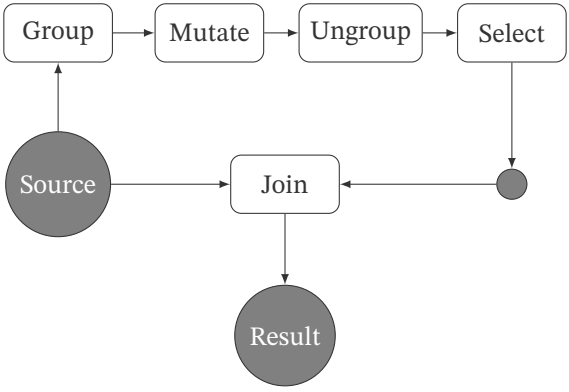
- Reduces the number of index columns
- Changes the observational unit
- Rows are grouped by the remaining indices
- **Not split-invariant**
  - Splitting may place parts of a group in different tables
  - Binding after projection violates the disjoint property
- Poor data schema + projection $\rightarrow$ incorrect conclusions

Grouped and arranged operations

When more flexibility is needed (e.g., preprocessing):

- **Aggregation function**: returns single value from tuple (sum, mean, count, first)
- **Window function**: returns tuple of same size (cumsum, lag, rank)
- Require specifying **groups** and/or **order**
- **Not split-invariant** (grouping is always required)

Mutating with groups and order



Example: `y = cumsum(x) group by category sort by date`

An algebra for data handling

Towards an algebra for data handling

- Formal algebra for data transformations is an open problem
- Statistical data handling $\neq$ relational algebra (transformation vs retrieval)
- Notable efforts: SDTA[1], LaraDB[2] (3 operators unifying RA + LA), Modin[3] (14 operators from pandas)
- Desirable properties: completeness, composability
- Split invariance is a novel concept proposed in this book
- Future: a "Turing complete" data handling language

[1] J. Song, H. V. Jagadish, and G. Alter (2021). **"SDTA: An Algebra for Statistical Data Transformation".** In: *Proc. of 33rd International Conference on Scientific and Statistical Database Management (SSDBM 2021)*. Tampa, FL, USA: Association for Computing Machinery, p. 12. DOI: 10.1145/3468791.3468811.

[2] D. Hutchison, B. Howe, and D. Suciu (2017). **"LaraDB: A Minimalist Kernel for Linear and Relational Algebra Computation".** In: *Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*. BeyondMR'17. Chicago, IL, USA: Association for Computing Machinery. ISBN: 9781450350198. DOI: 10.1145/3070607.3070608.

[3] D. Petersohn et al. (July 2020). **"Towards scalable dataframe systems".** In: *Proc. VLDB Endow.* 13.12, pp. 2033–2046. ISSN: 2150-8097. DOI: 10.14778/3407790.3407807.

Takeaways

- Split-invariant operations avoid sampling bias
- One must understand the properties and premises of the operations
- Index choice determines what conclusions are safe
- Projection/grouping is **not** split-invariant — use carefully

Questions?