Introduction to General Theory of Statistics

(Proposals for improving statistics and its methodological, information and software support in the context of new technologies and analytical capabilities presented by machine learning and AI methods.)

(Statistics, measurement, static and dynamic structure, adequacy criterion)

Statistics serves as a method for comprehending reality by reflecting the characteristics of the system being analyzed within a structured statistical information system (SIS). This representation is characterized by two key features: uniqueness and the retention of the system's parameter structure within the SIS, meaning the system's translation into the SIS should maintain a consistent structure, known as homomorphism. Exploring the fundamental attributes and the criteria for accurately representing the system within the SIS constitutes the core focus of general theory of statistics.

Concept of statistical measurement

Every theory relies on a formal language to communicate its ideas effectively. The general theory of statistics is no exception, and the language used must adhere to certain requirements dictated by the practical nature of statistics as a science. These requirements can be outlined as follows:

First and foremost, all elements of the language must be clearly defined in a constructive manner. This means that fundamental concepts in statistics such as
- statistical observation,
- the object of statistical observation,
- statistical indicators,
- statistical relationships, and
- systems of statistical indicators

must not only be named but also have their methods of construction provided.

Another term for this requirement is the operationality of all definitions.

Secondly, the key transformations on the elements of this language (such as the unification of concepts, transitioning from one concept to another, isolating a narrower concept from a broader one, etc.) should be defined unambiguously and have a clear interpretation in statistical terms.

Thirdly, the language must be both complete and consistent. This means that applying valid transformations to any elements of the language should not result in contradictions or the construction of a logical conclusion that cannot be interpreted in statistical terms.

The formal underpinning for the language of the general theory of statistics can be found in the mathematical **theory of measurements**. [1]

This theory, a specialized version of relational algebra, serves as a natural formalism for representing the logic underlying the general theory of statistics.

While the basic concepts and categories of measurement theory may not always align perfectly with their counterparts in statistics, the definition of "**measurement**" can be adapted to fit the statistical context. In statistical interpretation, measurement involves mapping an object of observation (such as a socio-economic system) into a system of statistical indicators. This mapping must be unambiguous and maintain the fundamental structural relationships of the observed system.

Statistical measurement, as a mapping of a socio-economic system into a system of statistical indicators, serves as the focal point of the general theory of statistics. This perspective, which emphasizes the process of obtaining statistical data through mapping, complements the traditional focus on the quantitative aspects of socio-economic phenomena.

Furthermore, by introducing the concepts of **statistical measurement** and **statistical scale**, the general theory of statistics can offer a more systematic and holistic approach to statistics as a science. The system of statistical indicators, comprising both initial and calculated data, is united by the purpose of statistical observation, which is determined by the consumers of statistical information.

The socio-economic system, as the object of statistical observation, is characterized by its complexity and interconnectedness. The integrity of the socio-economic system, along with its closed nature in terms of internal connections, underscores the challenge of analyzing and interpreting statistical data within such a system.

In order to ensure consistency, the system of statistical indicators must exhibit interdependencies and integrity. The relationships and connections between indicators reflect the structure of statistical information, which can be categorized into **static** and **dynamic** structures based on the nature of the connections between observation units and SES objects.

The **dynamic structure** of statistical information, which involves variable and causal relationships, is particularly important in understanding the evolving nature of socio-economic systems. Causal relationships, with their probabilistic underpinnings, play a crucial role in determining the dynamic structure of statistical information.

To address this aspect, we will draw upon P. Suppes' ideas regarding the probabilistic definition of causal relations. By incorporating Suppes' insights, we aim to provide a more robust and comprehensive framework for understanding the dynamic nature of statistical information and its implications for the general theory of statistics.

Formal definition of statistical information systems

The inherent discrete nature of statistical observations enables us to distinguish between **static** and **dynamic** structures of statistical information, a distinction we have previously mentioned.

Static structures capture the statistical information's current state, while **dynamic structures** encompass the connections that illustrate how each current state (or more precisely, the static structure at a particular moment) is influenced by previous states of statistical information. Thus, we conceive of **dynamic structures as the statistical counterpart to cause-and-effect relationships**.

At the core of the Statistical Information System (SIS) lie numerous statistical indicators. These indicators can be formally described as Composite Units of Information (CUI), a concept introduced by M.A. Korolev [10].

$$\mathbf{S.(P(1), P(2), P(3), \ldots, P(k-1), Q),} \qquad (1)$$

in which P(1), P(2), . . . , P(k-1) - correspond to the attribute attributes, and Q - to the basis attribute.

The **static structure** of the indicators reflects the associative connections between them, with some connections being established based on the analysis of the indicator descriptions.

For instance, if two indicators have the same value for the attribute "corporation," they are associatively related as they describe objects belonging to the same corporation. Similar associative connections can be identified through other attributes as well.

Another method of defining the static structure is through statistical groupings or clustering.

The compound unit of information (CUI) of a group of indicators is structured as a matrix, where the columns represent different attributes, and the rows denote the specific values of each attribute for various indicators. In simpler terms, attributes make up the columns, and indicators fill the rows.

When two statistical groups are said to be associated, it implies that their descriptions share components with identical names, and there exists at least one common value between the corresponding components of these groups.

To formally represent the associative aspect of the static structure of statistical information, the concept of a similarity or **tolerance** relation is utilized.

Within the context of a CUI, each attribute can give rise to its own tolerance relation. This specific relation is termed as **k-tolerance**, where 'k' signifies the index of the attribute and its associated tolerance relationship.

The **k-tolerance relation** is defined by its reflexive, symmetric, and intransitive properties and is depicted through a collection of labeled binary tolerance relations. Essentially, this showcases a scenario of multidimensional tolerance.

Moving on to the dynamic structure of the information system, it is viewed as a collection of cause-and-effect relationships among the system's elements. In developing algorithms to identify these relationships, the theories proposed by P. Suppes are taken as foundational.

P. Suppes' work primarily focuses on the analysis of cause-and-effect relationships among events, drawing upon the concept of events as defined in probability theory. Here, events are considered subsets within a predetermined probability space, characterized as instantaneous occurrences with their timing incorporated into their definitions.

Thus,

1. **P(At)** is the probability that event A will occur at time t,
2. **P(At|Bt')** - the probability that event A will occur at time t, provided that event B has already occurred at an earlier time t'.

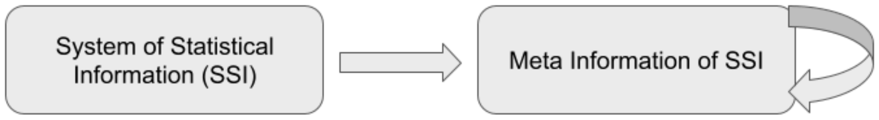
Definition of **root causes** (*prima facie*) in this case will look like this. Event Bt' is the root cause if the following conditions are met:

1. t' < t;
2. P(Bt') > 0;
3. P(At | Bt') > P(At).

We will use this concept in the definition of the dynamic structure of the SIS.

From statistical data to metadata

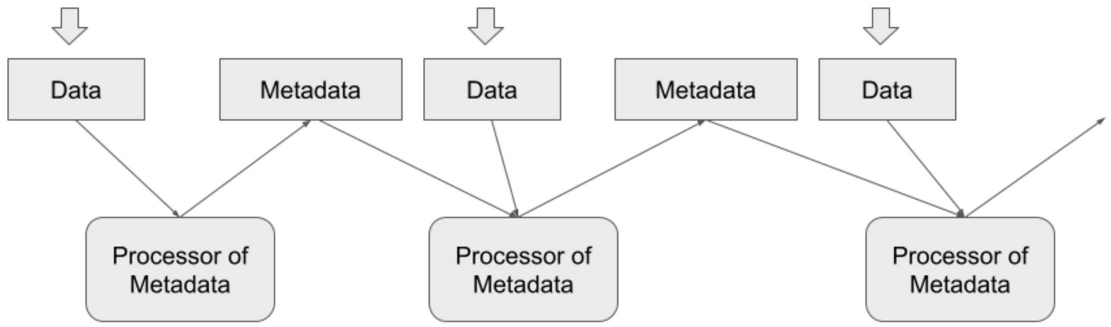
Metadata is essentially information about information. It encompasses any type of digital content that can be stored on a computer, including documents, databases, images, videos, audio files, and sensor signals. **From a metadata standpoint, all of these forms of data are treated equally and hold the same significance**.



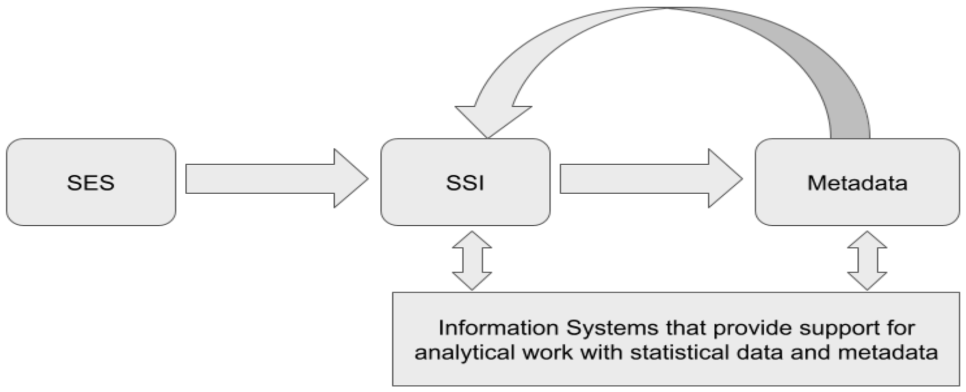
What, then, can be said about **the concept of metadata for metadata**? Essentially, metadata refers to data about data. **Thus, when we discuss metadata derived from metadata, we are essentially discussing the same entity**.

This point is crucial. **We propose to manage both the metadata of original data and the metadata of metadata through a singular metadata system**. This approach is visually represented in the figure, where we depict the metadata loop closing in on itself.

Furthermore, we delineate the ongoing process of generating metadata, which evolves over time both from the initial data and from metadata previously created. This cyclical process highlights the dynamic and iterative nature of metadata generation.



By integrating socio-economic systems (SES) mapping into statistical information systems (SIS) through statistical observation, and then mapping SIS into Metadata, we can develop a comprehensive and generalized scheme.


SES
SSI
Metadata
Information Systems that provide support for analytical work with statistical data and metadata

The diagram illustrates the SES → SIS and SIS → (Metadata), clearly demonstrating how it maintains the integrity and structural relationships within the displayed system, as discussed earlier.

This approach not only highlights the crucial characteristic of statistics as a system— its closed nature in relation to the Statistical Information System (SIS)—but also ensures that **any data**

encompassed by or generated within the system throughout its processing phase is seamlessly integrated into the SIS. This integration process is precise, safeguarding the original data's structure.

In the contemporary landscape, statistical information transcends traditional statistical indicators and tables. It leverages an array of computer technologies for presenting statistical data and the outcomes of statistical analyses. This encompasses everything from basic tables and charts to sophisticated multimedia and virtual reality presentations.

When it comes to metadata, it encompasses all forms of data. For these datasets, it's imperative to create descriptive metadata elements and to forge connections among these elements.

Conceptually, metadata can be visualized as a graph. Within this graph, metadata elements are depicted as nodes, while the links between these elements are represented by the graph's edges. This structure facilitates a comprehensive and interconnected representation of metadata, enhancing the understanding and utilization of statistical information.

Graph Database for Metadata Representation

The graph can be defined as follows:

$$G = \{V, E\},$$

Where

G - graph;
V - is the set of graph nodes, which in the case of statistical information systems will represent composite units of information (CUI);
E - is the set of edges connecting connected nodes of the graph.

Node $v \in$ **V,** as we noted, is a CUI and can be described as **struct** in programming languages C++, Rust, Julia or **Dict** in Python language. In the code snippet below, we have used Julia notation.

Python
```
struct Node <: AbstractGraphType
        sha1::String
        labels::Vector{String}
        d_sha1::String
        card::Int
        dataset::Vector{Int}
        props::Config
 end
```

The above structure can be considered as a natural extension of the CUI, in which we have supplemented each attribute with an indication of its type.**This structure is standard for representing any graph node**. The set of CUI details is minimal and serves to ensure unambiguous identification of each node of the graph.

In this structure we distinguish three parts:

1. The external description of the node, which, in turn, is represented by two attributes:
    a. **sha1** - unique identifier (SHA1 hash);
    b. **labels** - labels (one or more) that reflect the semantics of a given node;
2. Internal description of a node, which describes the content of the CUI represented by this node. Attributes of this description include:
    a. **d_sha1** - SHA1 hash generated from the contents of the SEI;
    b. **dataset** - vectorized representation of the contents of the CUI (we will dwell on the vector representation of the CUI in more detail in the next section);
    c. **card** - the number of non-repeating CUI elements;
3. **props** - additional node attributes, presented as a JSON structure.

The edges of the graph describe the connections between nodes. Any pair of nodes can have more than one edge, and each edge has a direction.

```Python
struct Edge <: AbstractGraphType
        source::String
        target::String
        r_type::String
        props::Config
end
```

Edge description attributes:

1. **‘source’** - sha1 identifier of the starting node;
2. **‘target’** - sha1 end node identifier;
3. **‘r_type’** - edge label reflecting the type of connection between nodes;
4. **‘props’** - additional edge attributes, presented as a JSON structure.

The SIS vocabulary, or SIS dictionary, encompasses all possible values for attributes. This collection of terms is dynamic, constantly growing to incorporate new concepts and terms as the observed population and SIS evolve.

To describe the SIS dictionary we use the following structure:

```Python
struct Token <: AbstractGraphType
        id::You
        bin::Int
        zeros::Int
        token::Set{String}
        tf::Int
        refs::Set{String}
end
```

In this structure:

- `**id**`: An integer generated by hashing the value of the word (token), where a word refers to any code that signifies a basic unit of data, irrespective of its data type.
- `**bin**`: The first k bits of the hash code, indicating the word's position in the vector representation of the CUI (explained further in the following section).
- `**zeros**`: The count of consecutive zeros observed at the end of the hash's bit representation.
- `**tf**`: The term frequency, which is how often a particular word occurs within the CUI.
- `**refs**`: A collection of sha1 identifiers representing all the CUIs that include this word.

These three structures serve as the foundation for the graph representation of the statistical information system (SIS).

HllSet as a way to approximate the contents of CUI

The CUI approximation algorithm draws upon the HyperLogLog algorithm, a renowned method for probabilistically estimating the count of unique elements within vast datasets.

To understand the HyperLogLog algorithm, it's essential to grasp its underlying principle. This algorithm estimates the magnitude of a set containing uniformly distributed random numbers by analyzing the longest sequence of leading zeros in their binary representations. For instance, if the longest sequence of leading zeros found among these numbers is 'n,' the algorithm estimates the set's unique element count to be 2^n [6].

In practice, the HyperLogLog algorithm enhances the accuracy of this estimation by first applying a hash function to each item in the original set. This process transforms the set into a new collection of uniformly distributed random numbers, maintaining the original set's size. The unique element count of this transformed set is then estimated using the maximum leading zeros method.

However, this straightforward estimation approach can suffer from high variance. To address this issue, the HyperLogLog algorithm segments the transformed set into several subsets. It then calculates the longest sequence of leading zeros within each subset. By averaging these calculations using the harmonic mean, the algorithm produces a more accurate and less variable estimate of the total set's unique element count.[7]

HllSet is a method for encoding datasets of any type and virtually any size. It is based on the HyperLogLog algorithm, with modifications made to the register's structure. These adjustments have demonstrated that HllSet is a suitable representation of the original datasets and satisfies all properties of set theory.

If you are curious about the technical intricacies of encoding datasets into HllSets, you can delve deeper into the algorithm by checking out the references [8, 9]. The outcome of this process we call as the HllSet, short for HyperLogLog Set.

As described in the original algorithm, the size of HllSet remains constant and is independent of the size of the source data. This means that whether a CUI represents a single indicator or a grouping of thousands of indicators, the size of the HllSet will be the same. The size of the HllSet is solely determined by the number of subsets (registers) used to encrypt the data. For 64-bit processors, where an integer is 8 bytes, the size of the HllSet can be calculated using the following formula:

$$\mathbf{M = 8 \times \| S \|},$$

Where

$\| S \|$ - the number of subsets that registers represent.

The HllSets define all standard set operations, meeting all the requirements of set theory.

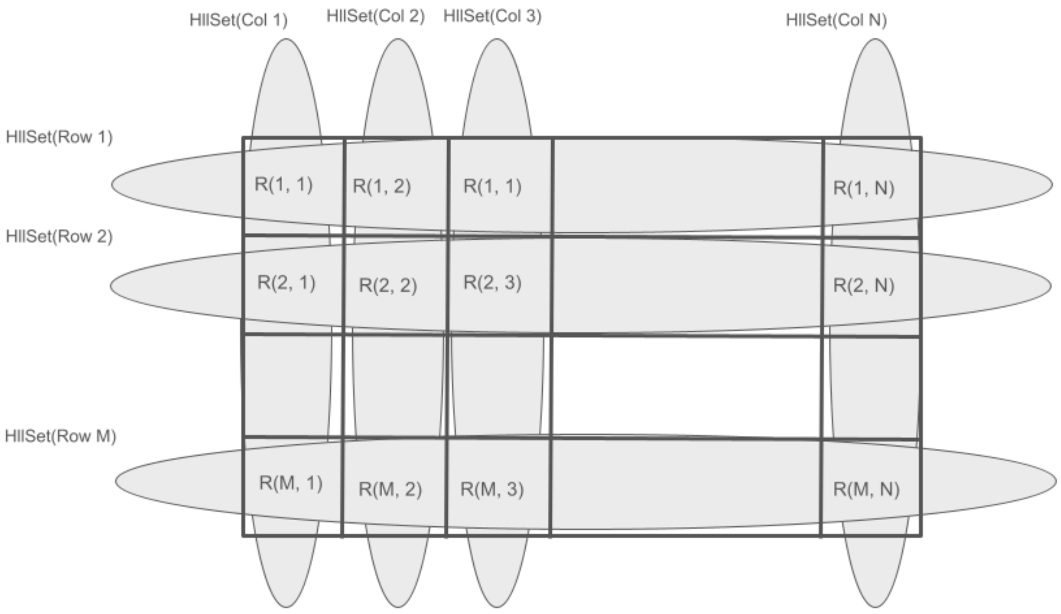
```
Unset
 Z - empty set
 U is a universal set, in practical applications it is the union of
all HllSet defined in a specific application

 1. (A ∪ B) = (B ∪ A): true                    // commutativity
 2. (A ∩ B) = (B ∩ A): true
 3. (A ∪ B) ∪ C) = (A ∪ (B ∪ C)): true           // associativity
 4. (A ∩ B) ∩ C) = (A ∩ (B ∩ C)): true
 5. ((A ∪ B) ∩ C) = (A ∩ C) ∪ (B ∩ C): true  // distributivnost'
 6. ((A ∩ B) ∪ C) = (A ∪ C) ∩ (B ∪ C): true
 7. (A ∪ WITH) = A: true                           // identity
 8. (A ∩ IN) = A: true
 9. (A ∪ A) = A: true
10. (A ∩ IN) = A: true
```

Using HllSet to Present Tabular Data

Formally, tabular data sets can be described as a subset of the Cartesian product of two sets:

$$R(S_1, S_2) \subseteq S_1 \times S_2$$

However, in the scenario of HllSet, the direct access to set elements is not available. Therefore, an alternative method must be employed in order to construct and utilize table structures.



The illustration above demonstrates how binary relationships can be created between HllSets. For instance, when dealing with CSV files, the process would involve the following:

- **S_1** - a collection of HllSets representing lines from the CSV file
- **S_2** - a collection of HllSets representing columns CSV file.

Each cell in the table view represents the intersection of an HllSet row and an HllSet column, resulting in a new HllSet.

All elements within an HllSet are part of a common SIS dictionary (tokens) and are assigned the same identifiers used during the construction of the HllSet. This allows for easy tracing of the contents back to their corresponding tokens in the dictionary, enabling the restoration of cell contents.

When working with HllSets, the following considerations should be taken into account:

- The contents of matrix cells consist of an unordered collection of tokens from the original CSV cell. To determine the order of these tokens, reference to the original CSV file is necessary.
- In many instances, only a subset of rows are utilized, which must be kept in mind when working with the data.

The correction process is typically straightforward:

- To obtain the actual HllSet for a column, the union of the HllSets of the cells within that column must be used.
- The same approach applies to working with HllSets.

An example of using metadata for searching in SIS

We'll look at two examples of using metadata:

1. Recovering tabular data from CSV files;
2. Search for CSV files and columns that match words from the search query.

Recovering tabular data

Python
```python
'''
First we need to generate metadata for the columns and rows of the files we are going to work with.
During proceSISng, we will add metadata to two main tables of the Graph Database:
- nodes;
- tokens.
In the nodes table we will place row and column data for each file. Each row and column will receive a unique SHA1 identifier.
All new words from the files will be entered into the tokens table, indicating the SHA1 identifiers of the rows or columns of the files from which these words were selected.
This will establish a relationship between the tokens (words) and the elements of the files they were in.

db - Graph Database;
row.name() is a function that retrieves the full filename from the row object that represents the file.
'''
for row in eachrow(file_list)
    Store.ingest_csv_by_row(db, row.name())
```

```
end

for row in eachrow(file_list)
    Store.ingest_csv_by_column(db, row.name())
end
```

The Graph Database (Store object) has specialized methods for extracting data matrices from the intersections of rows and columns of CSV files.

```
Python
"""
    Here we are going to extract the row and column nodes from the csv file.
    The resulting matrix will show the number of elements at the intersection
of the row and column nodes.
"""
matrix = Store.get_card_matrix(db, source_id)
for row in each row(matrix)
    println(row)
end
```

Output after executing this program:

```
Unset
[2.0, 2.0, 2.0, 1.0, 2.0, 4.0, 3.0, 3.0]
[2.0, 2.0, 1.0, 1.0, 2.0, 4.0, 5.0, 4.0]
[2.0, 3.0, 2.0, 1.0, 1.0, 4.0, 4.0, 4.0]
[2.0, 2.0, 2.0, 1.0, 1.0, 4.0, 4.0, 3.0]
[2.0, 2.0, 2.0, 1.0, 2.0, 4.0, 3.0, 3.0]
[2.0, 3.0, 2.0, 1.0, 1.0, 4.0, 3.0, 3.0]
[2.0, 2.0, 2.0, 1.0, 2.0, 4.0, 3.0, 3.0]
[2.0, 3.0, 2.0, 1.0, 2.0, 4.0, 3.0, 3.0]
[2.0, 2.0, 1.0, 1.0, 2.0, 4.0, 3.0, 3.0]
. . .
```

As we see, many matrix cells have several elements.

```
Python
matrix = Store.get_value_matrix(db, source_id)
for row in eachrow(matrix)
    println(row)
end
```

And this is the output from this program (only the first few lines of output are shown):

```
Python
["[\"Serious\"]", "[\"Pedestrian\"]", "[\"Male\"]", "[]", "[\"Friday\"]",
"[\"Kensington\",\"Chelsea\",\"and\"]", "[\"Motorcycle\",\"over\",\"and\"]",
"[\"Not\",\"Pedestrian\",\"pedestrian\"]"]
["[\"Serious\"]", "[\"Pedestrian\"]", "[\"Female\"]", "[]", "[\"Wednesday\"]",
"[\"Kensington\",\"Chelsea\",\"and\"]",
"[\"car\",\"Taxi/Private\",\"and\",\"hire\"]",
"[\"croSISng\",\"Pedestrian\",\"CroSISng\",\"ped.\",\"facility\"]"]
["[\"Serious\"]", "[\"rider\",\"Driver\"]", "[\"Male\"]", "[]", "[\"Monday\"]",
"[\"Kensington\",\"Chelsea\",\"and\"]",
"[\"cycle\",\"Motor\",\"and\",\"under\"]",
"[\"croSISng\",\"carriageway\",\"elsewhere\"]"]
["[\"Serious\"]", "[\"Pedestrian\"]", "[\"Male\"]", "[]", "[\"Monday\"]",
"[\"Kensington\",\"Chelsea\",\"and\"]",
"[\"cycle\",\"Motor\",\"and\",\"under\"]",
"[\"Not\",\"Pedestrian\",\"pedestrian\"]"]
["[\"Serious\"]", "[\"Pedestrian\"]", "[\"Male\"]", "[]", "[\"Sunday\"]",
"[\"Kensington\",\"Chelsea\",\"and\"]", "[\"Car\",\"and\"]",
"[\"Not\",\"Pedestrian\",\"pedestrian\"]"]
["[\"Serious\"]", "[\"rider\",\"Driver\"]", "[\"Male\"]", "[]",
"[\"Tuesday\"]", "[\"Kensington\",\"Chelsea\",\"and\"]",
"[\"Motorcycle\",\"over\",\"and\"]", "[\"Not\",\"pedestrian\"]"]
. . .
```

Finding nodes in a graph containing words from a query

In this scenario, we utilized Neo4J Graph as the server for our Graph Database. To interact with Neo4J, we employed a custom-built interface called LisaNeo4J. This interface offers essential functionalities for graph operations, such as data retrieval.

The following snippet illustrates a program using simplified Julia code to outline the fundamental procedures for handling a request.

```
Python
# Find nodes and edges (SHA1 identifiers) that contain "sex", "taxi" and "day"
rows = LisaNeo4j.search_by_tokens(db.sqlitedb, "sex", "taxi", "day")
# Fetch all edge references
edges = Vector()
edges_refs = LisaNeo4j.select_edges(db.sqlitedb, rows, edges)

# Fetch all links to nodes
nodes = Vector()
LisaNeo4j.select_nodes(db.sqlitedb, refs, nodes)

# Building a presentation in Neo4J
#1. Let's start with nodes
for node in nodes
    labels = replace(string(node.labels), ";" => "")
    query = LisaNeo4j.add_neo4j_node(labels, node)
    data = LisaNeo4j.request(url, headers, query)
end
# 2. After this, connect the nodes with edges
for edge in edges
    query = LisaNeo4j.add_neo4j_edge(edge)
    data = LisaNeo4j.request(url, headers, query)
end
```

Search results enable us to conduct fundamental analytics. In this instance, we are aiming to assess the connections between CSV files that meet our specified query criteria. We compute two key metrics: Jaccard distance and Cosine similarity.

```
Python
# Defining a Cypher request
query = LisaNeo4j.cypher("MATCH (n:csv_file) RETURN n.labels, n.sha1, n.d_sha1,
n.dataset, n.props LIMIT 20")

hlls = LisaNeo4j.collect_hll_sets(query, hlls)
# Define relationships between related csv files
for (k, in) in hlls
  for (d1, v1) in hlls
    if k != k1
      jaccard = SetCore.jaccard(v.hll_set, v1.hll_set) # jaccard comparison
       println("jaccard: ", jaccard)
      cosine = SetCore.cosine(v.hll_set, v1.hll_set)    # cosine comparison
      println("cosine: ", cosine)
    end
  end
end
```

And this is the conclusion from this program.

```
Unset
jaccard: 78
cosine: 87.0
jaccard: 78
cosine: 87.0
```

In our example, we utilized 2 files, resulting in one relationship being measured by two different methods. It is important to highlight that we obtained two relationships for each metric due to the directed nature of our graph. In directed graphs, symmetric connections are depicted by two edges: one from the first node to the second, and another from the second to the first. Additionally, there is a notable discrepancy in the values of these two metrics.

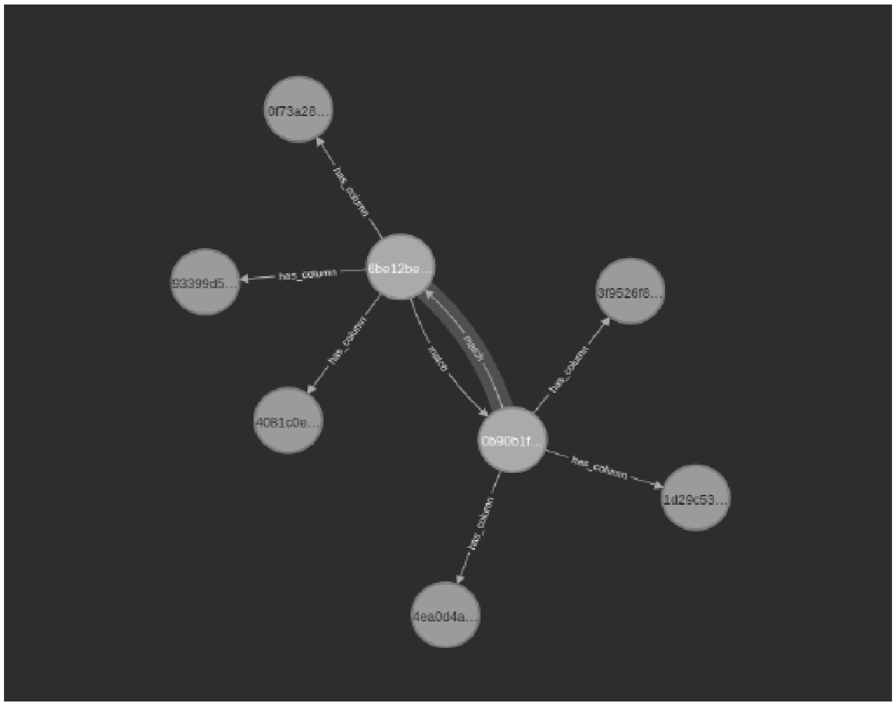


Illustration of the created graph in the Neo4J browser:

For access to the complete programs and extra documentation, please visit the following link.[9]

Summary

The document presents an advanced exploration of statistical theory, focusing on improving statistical methods, information, and software support through new technologies like machine learning and AI. It delves into the core principles of statistical measurement, the structured representation of socio-economic systems (SES) within Statistical Information Systems (SIS), and the formal definitions of statistical information systems, highlighting both static and dynamic structures.

Key points include:
- The importance of a formal language in statistical theory, emphasizing clear, constructive definitions and operationality for concepts such as statistical observation and indicators.
- The adaptation of measurement theory to fit statistical contexts, where measurement is seen as mapping an observed system into a system of statistical indicators, maintaining the system's fundamental structural relationships.
- The discussion on static and dynamic structures within SIS, with dynamic structures capturing the evolving nature of socio-economic systems through cause-and-effect relationships.
- The introduction of Composite Units of Information (CUI) for formally describing statistical indicators and the use of graph databases to represent and manage metadata, enhancing the understanding and utilization of statistical information.
- The adoption of the HyperLogLog algorithm for approximating the contents of CUI, demonstrating its practical application in encoding datasets of any size.
- Practical examples of generating and utilizing metadata for searching within SIS, including the recovery of tabular data from CSV files and searching for specific information through graph databases.

Overall, the document outlines a comprehensive approach to modernizing statistical theory and practice by leveraging new technologies and methodologies. It emphasizes the importance of maintaining the integrity and structure of statistical information while exploring innovative ways to analyze and interpret complex socio-economic systems.

References

1. Johann Pfanzagl. Theory of Measurement. Springer-Verlag Berlin Heidelberg 1971
2. https://durham-repository.worktribe.com/preview/1421323/17400.pdf
3. https://ru.wikipedia.org/wiki/Отношение_толерантности
4. Suppes: Scientific Philosopher, Volume 1. Probability and Probabilistic Causality, ed. P. Humphreys. 339-366. Dordrecht: Springer.
5. https://en.wikipedia.org/wiki/HyperLogLog
6. https://algo.inria.fr/flajolet/Publications/FlFuGaMe07.pdf

7. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/40671.pdf
8. https://github.com/alexmy21/lisa/blob/main/README.md
9. https://github.com/alexmy21/lisa_meta
10. https://fileskachat.com/download/33578_259e926ae6cf029d720a255d68658772.html