SmartDataLake

DELIVERABLE D4.2

Initial version of the visual analytics engine

PROJECT NUMBER: 825041
START DATE OF PROJECT: 01/01/2019
DURATION: 36 months

Horizon 2020

SmartDataLake is a Research and Innovation action funded by the Horizon 2020 Framework Programme of the European Union.

Dissemination Level	Public
Due Date of Deliverable	Month 18 (30/06/2020)
Actual Submission Date	22/06/2020
Work Package	WP4: Scalable and Interactive Visual Analytics
Tasks	Task 4.2: Feature exploration and parameter tuning
Type	Other
Lead Beneficiary	University of Konstanz
Approval Status	Submitted for approval
Version	1.0
Number of Pages	40
Filename	SmartDataLake-D4.2-Initial-version-of-the-visual-analytics-engine.pdf

Abstract

This document presents the initial version of the visual analytics layer of SmartDataLake. It describes user-interfaces, visualizations, and interaction techniques for analysis tasks in different stages of the SmartDataLake analysis pipeline, as well as the background computations and interfaces that are needed to provide the desired functionalities.

History

Version	Date	Reason	Revised by
0.1	04/06/2020	Initial Version	Thilo Spinner
0.2	17/06/2020	Revision by SYNYO	Thomas Paulin
0.3	18/06/2020	Revised version	Thilo Spinner
1.0	22/06/2020	Final version for submission	Dimitris Skoutas

Author List

Organization	Name	Contact information
UKON	Thilo Spinner	thilo.spinner@uni-konstanz.de
ARC	Georgios Chatzigeorgakidis	gchatzi@athenarc.gr
ARC	Dimitris Skoutas	dskoutas@athenarc.gr
TU/e	Hamid Shahrivari Joghan	h.shahrivari.joghan@tue.nl
SYNYO	Thomas Paulin	thomas.paulin@synyo.com

Executive Summary

This deliverable summarizes the current state of the visualization layer of SmartDataLake, SDL-Vis. It describes how the concepts and techniques defined in D4.1, "Interactive visual analytics model," are instantiated in a system implementation. Since the frontend application of SDL-Vis, the Visual Explorer, and the backend application of SDL-Vis, the Visual Analytics Engine, are tightly coupled, the implemented functionalities are described as a communication and interaction process between visualization frontend, visualization backend, and the respective lower-level components of SmartDataLake. During this communication process, data from the data lake and access protocols are increasingly abstracted with each layer. This document breaks down in detail how the communication and data processing protocols are instantiated in the components of SDL-Vis and gives an overview of the used technologies and the implementational details of the Visual Explorer and the Visual Analytics Engine application.

Section 1 gives an introduction to the general architecture of the SDL-Vis applications and anchors SDL-Vis in the overall project context of SmartDataLake. The purposes and functionalities of the two main components of SDL-Vis, the Visual Analytics Engine and the Visual Explorer, are described in subsections 1.1 and 1.2, respectively. Subsection 1.3 sets the content of this document, and SDL-Vis in general, into relation to all other related work packages and tasks of SmartDataLake. Setting the applications of SDL-Vis into context with the full SmartDataLake analysis pipeline is particularly relevant since the functionalities and design decisions for the implementation are based on the points where the visualization layer plugs into specific stages and components of the pipeline.

Section 2 provides an overview of the software architecture of the Visual Explorer (2.1) and the Visual Analytics Engine (2.2) applications. While describing the used technologies, programming languages, and libraries, the chapter points out how specific design goals and requirements, such as responsive user-interfaces and the reusability of components, are realized in the implementation.

Following the description of the toolchain, Section 3 covers the implemented functionalities of SDL-Vis. Since the visualizations and interaction mechanisms of the visualization layer are highly task-driven, each of the subsections 3.1 to 3.6 covers one specific analysis task in the SmartDataLake analysis pipeline. Therefore, each subsection includes a description of the task itself while explaining the corresponding views, visualizations, and user interactions of the Visual Explorer. In parallel, the backend interfaces and mechanisms of the Visual Analytics Engine that enable these functionalities are described.

In section 4, a final conclusion summarizes the current state of the implementation and gives an outlook on the upcoming tasks and the respective visual analytics components.

Abbreviations and Acronyms

API	Application Programming Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
HIN	Heterogeneous Information Network
HTML	Hypertext Markup Language
IT	Information Technology
JSON	JavaScript Object Notation
REST	Representational State Transfer
SDL	SmartDataLake
SQL	Structured Query Language
t-SNE	T-distributed stochastic neighbor embedding
UI	User Interface
VA	Visual Analytics
WP	Work Package

Table of Contents

1. Introduction

The vast amount of data being collected and stored in modern information technology systems poses new challenges to the analysis of such data. Since a human's cognitive and analytical capacity is limited, automated algorithms are useful tools to process the data and discover meaningful connections in it. Besides the sheer amount of collected data, the variety of different data formats and types further complicates the analysis. Even modern mining and machine learning techniques rely on the meaningful preparation of data, often leading to complex data loading and pre-processing pipelines. Complex algorithms then strive to detect meaningful connections between data sources and entities, enabling the human to make sense of the data collection and generate new knowledge.

Every component in this multi-stage data processing pipeline is dependent on various design decisions. Inspection and combination of different data sources, data annotation, data enrichment, as well as the parameter choices for mining algorithms involve complex human decisions and interactions. In return, these decisions critically influence the results of the data processing pipeline, meaning that inappropriate or uninformed parameter choices might lead to no or misguided insights in the data.

Visual Analytics tackles this problem by combining the strengths of human and machine in an iterative analysis cycle. Complementary to presenting the results of automated models to the user, the models themselves are analyzed. Since the models significantly influence the results, understanding the full data processing and mining pipeline is the foundation for trustworthy decision making. By inspecting models and results simultaneously, the influence of abstract parameter choices can be observed and interpreted by the human, enabling him to refine the parameters and, therefore, steer the models according to his knowledge.

As shown in Figure 1, the SmartDataLake data analysis pipeline involves a variety of different data loading, data processing, and data mining stages, all of them being dependent on human decisions and parameter choices. Furthermore, the generated results of each intermediate processing step are complex to interpret and differ in data format and type, rendering task-driven visualization and interaction techniques vital. Therefore, visual analytics plays a crucial role in the project to enable human understanding and refinement of the involved models, parameters, and results.

Dependent on task and component, the visual analytics layer (SDL-Vis) has to fulfill different requirements. While preserving the real-time capabilities of the visualization and interaction frontend, vast amounts of data have to be queried, processed, cached and presented to the user. Furthermore, viewport and model state have to be maintained per user-session, enabling the execution of multiple user-specific analysis workflows at the same time.

Data Querying — To plug into the different stages of the SmartDataLake analysis pipeline, the visualization layer has to query and receive data in various formats and types, as summarized in Table 1. Therefore, the visualization layer has to implement different communication protocols to interface with all components of SmartDataLake.

Component	Query Type	Response Type
Data Virtualization	SQL	Tabular Data Database Schema
Approximate Query Processing	SQL	Tabular Data Uncertainty
Entity Resolution & Ranking	JSON Object G-Core	Tabular Node-Link Graph Database Schema
Similarity Search	JSON Object	Tabular Data Similarity Matrix
Link Prediction[1]	*JSON Object* *G-Core*	*Tabular Node-Link* *Graph Database Schema*
Community Detection[1]	*JSON Object* *G-Core*	*Tabular Node-Link* *Graph Database Schema*

Table 1: The formats of queries and responses for the different components of SmartDataLake, as depicted in Figure 1.

Data Processing – The data returned from the components of the SmartDataLake analysis pipeline needs to be further transformed, filtered, and converted to a standardized format priorly to feeding it into the visualization component. The data transformation task includes aggregation based on statistical descriptors, dimensionality reduction using different projection techniques, as well as data augmentation. Filters help to reduce the amount of data presented to the user based on the viewport of the visualization component or user-defined constraints. Finally, the data has to be converted to a format that can be processed by the visualization and interaction frontend. This includes the conversion of graphs into a node-link property graph representation, as well as the encoding of relational data and database schemas.

Data Storage – To maintain the real-time capabilities of the visualization frontend and avoid redundant computations, data has to be cached by the system. This concerns the intermediate storage of data, but also more complicated mechanisms, such as speculative execution [1]. By pre-computing time-consuming operations that are likely to be applied by the user in the future, results for future actions can be accessed from the cache of the system, enabling real-time user-feedback for otherwise non-real-time operations. Besides the reduction of access times, storing a history of results is essential to identify and communicate changes in models or data. For the analyst, change indication is critical to assess how a parameter refinement influences models and results. Finally, session management is an essential sub-task of the visual analytics layer. The visualization system has to identify user-sessions to allow multiple users to work on different data

[1] Upcoming Task

and model states. Therefore, besides communicating with the components of the SmartDataLake data processing pipeline, the visualization system has to maintain a persistent configuration for all components that are dependent on additional parameters.

Data Representation – Finally, the processed and converted data has to be visually presented to the user. While the analysis of raw data or models might be possible for simple analysis tasks, visualizations help to abstract more complex data in a form that can be more easily perceived and interpreted by humans. Furthermore, visualizations allow intuitive interaction with data and models, for example, by adapting model parameters through viewport modifications. By jointly visualizing models and data, abstract parameter choices can be made accessible by the user, and the influence of parameter refinements can be directly encoded in the visual representation.

The above tasks can be divided into two main aspects: **data provisioning**, covering data querying, data processing, and data storage, as well as **visual data representation**, including data visualization and interaction. This allows us to split the visual analytics layer into two distinct applications, the **Visual Analytics Engine**, being responsible for data provisioning, and the **Visual Explorer**, covering visual data representation and user interfaces.

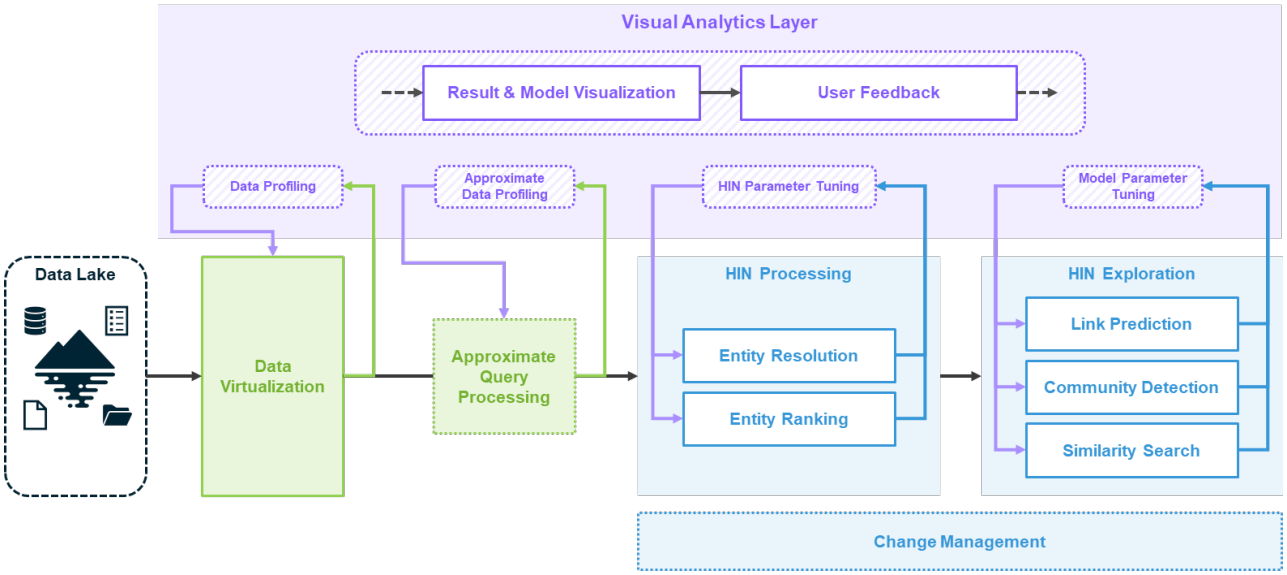

Figure 1: The data analysis pipeline of SmartDataLake. Visual analytics is relevant during all stages of the pipeline. Therefore, the visual analytics engine plugs into the intermediate results of each sub-component, enabling data-, model-, and parameter analysis and refinement over the full data processing pipeline.

1.1. The Visual Analytics Engine in SDL-Vis

As the backend application of SDL-Vis, the **Visual Analytics Engine** builds the bridge between lower-level components of the SmartDataLake processing pipeline and the visualization frontend, Visual Explorer. It interfaces with these components and provides abstract access to their data, models, and results, as depicted in Figure 2.

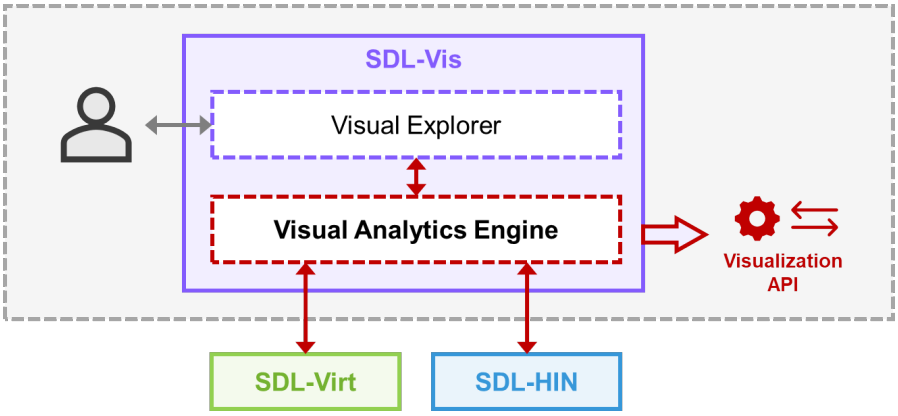

Figure 2: The Visual Analytics engine builds the interface between the lower-level components of SmartDataLake and the Visual Explorer, exposing its functionality via the visualization REST API.

Each stage of the processing pipeline, as well as its intermediate results, are relevant for the SmartDataLake visual analytics process. Starting from raw data and schema information on the data foundation, the Visual Analytics Engine directly connects to the data virtualization component SDL-Virt. Analyzing the heterogeneous data sources is essential for an informed analysis since the quality and structure of the data have a fundamental impact on the results of the automated mining algorithms and the connections that can be discovered in the data.

Automated models are highly useful to make huge data collections accessible and generate human knowledge from them. By directly connecting to the single models and algorithms of the data mining component SDL-HIN, the Visual Analytics Engine can query results and update user-steered parameters of each individual model. To enable the task-driven analysis of automated models, the Visual Analytics Engine defines custom data processing and transformation workflows for each of them. The functionality, which is exposed under different REST API endpoints, corresponds directly with the user-interfaces, visualizations, and interactions offered in the visual analytics frontend Visual Explorer.

Figure 1 shows how the visualization layer plugs into different stages of the SmartDataLake analysis pipeline, enabling visual analytics on each of the involved components. In this pipeline, the Visual Analytics Engine implements the backend interfaces for the results and model visualization, as well as an abstraction of the iterative model refinement functionality via user-feedback.

1.2. The Visual Explorer in SDL-Vis

The **Visual Explorer** is the user-interfacing component of SDL-Vis. It includes task-driven visualizations for data, models, and results of the SmartDataLake processing pipeline. By exposing data, parameters, and functionalities of the involved models in a consolidating application, it builds the primary interface between the analyst and the SmartDataLake processing pipeline.

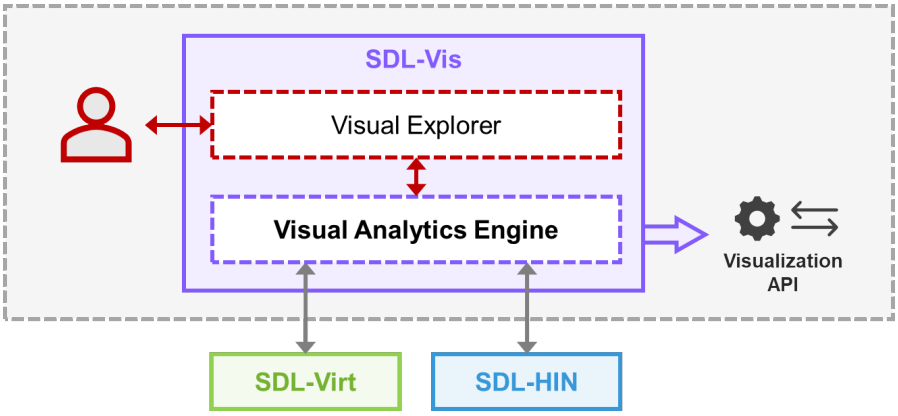

Figure 3: The Visual Explorer is the user-interfacing component of SDL-Vis. It provides task-driven visualizations and interaction techniques to analyze data, models, and results of the SmartDataLake processing pipeline.

The user-interface of the Visual Explorer is modularized into multiple **visualization and interaction panels**. Each panel addresses one task and plugs into a specific stage of the analysis workflow, as shown in Figure 1. Based on the functionality of the Visual Analytics Engine, it requests data over the provided REST API. Parameters and results are visualized, supporting human knowledge generation. Necessary data or model updates can be derived from the analyst's insights and fed back into the analysis pipeline. While the distinct visualization and interaction panels typically work on a specific sub-task or sub-component of SmartDataLake, they all operate on the same data foundation. This enables the gradual refinement of the different model stages involved in the complex data mining pipeline, without having intermediate results interfering with each other and, therefore, falsifying the insights derived from the analysis process.

The different analysis tasks and data mining models place highly specific requirements on the visualization and interaction design. Therefore, the Visual Analytics Engine provides pre-formatted and transformed data to support the distinct visualization and interaction panels of the Visual Explorer. This keeps computationally and storage-intensive operations separated from the user-interface, preserving real-time analysis of models and results. At the same time, background operations can be issued to the lower-level components of SmartDataLake, preparing and pre-computing results for possible future user-actions.

1.3. Relation to other Work Packages

The Visual Analytics Engine and the Visual Explorer are the core elements of SDL-Vis, the interactive visual analytics component of SmartDataLake. The Visual Analytics Engine provides access to the lower-level components of SmartDataLake, i.e., SDL-Virt and SDL-HIN, while the Visual Explorer covers user-interfaces, results visualization and interactive visual analytics on the components of SmartDataLake. The interplay between all elements of SmartDataLake to enable visual analytics during all stages of the data analysis pipeline is defined in Deliverable D4.1, "Interactive visual analytics model." The Visual Analytics Model builds the theoretical foundation for the implementation of SDL-Vis and describes the relevant interactions between the user and the automated parts of the analysis pipeline. For each component, it defines task-driven visualization and interaction panels, which provide visual representations of models and data, enabling an iterative knowledge generation and model refinement loop, as shown in Figure 4. The Visual Analytics Engine is implemented to facilitate the functionalities described in the Visual Analytics Model and provide the necessary bridges between lower-level components and the Visual Explorer. Therefore, its design is mostly driven by the different visualization and interaction panels, which enable Visual Analytics on the distinct sub-tasks of the SmartDataLake project.

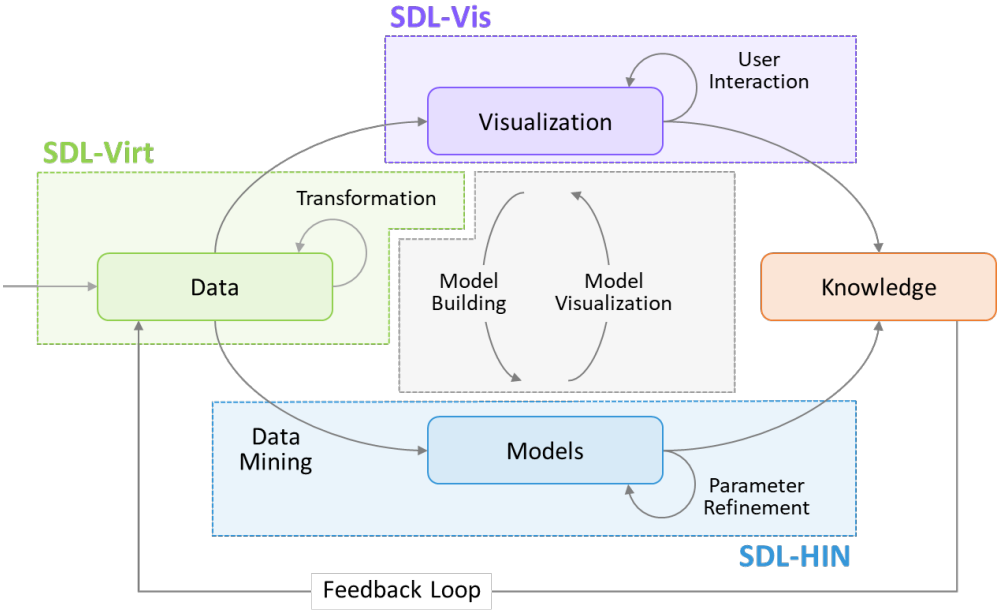

Figure 4: The Visual Analytics Process in SmartDataLake [7]. Knowledge generation is based on interactions between data, models, visualizations and the user.

1.3.1. Tasks of SmartDataLake

Beginning with the data provisioning layer SDL-Virt, a critical use-case in SmartDataLake is data profiling. To assess the quality and structure of the data foundation, the analyst has to inspect schema information, examine available attributes, find missing values, and apply descriptive statistics to the dataset. This task has to happen right at the start of the processing pipeline before

applying more complex models, such as pattern mining or data augmentation algorithms. Therefore, the Visual Analytics Engine has to connect directly to the data virtualization layer, as defined in D2.1, "Query engine over virtualized data." This layer provides abstract access to heterogeneous data sources of SmartDataLake, exposing them through a structured query language (SQL) in the form of relational data. Building on the data virtualization layer, the approximate query engine, as defined in D2.2, "Data synopses for approximate analytics," allows getting approximate aggregated results from the database layer. This is especially useful for the data profiling task since, for an interactive exploration of the data foundation, fast results are more important than precision. For example, the overall shape of a frequency distribution can still be observed from a histogram where the individual bar sizes might vary with a predefined accuracy.

Regarding the information mining layer SDL-HIN, a series of different components and analysis tasks set distinct requirements on the Visual Analytics Engine. In general, the Visual Analytics Engine abstracts the data access for these tasks and transforms the data accordingly. As part of D3.1 and D3.2, "Similarity search over entity profiles" specifically involves results transformation and speculative model execution, handled by the Visual Analytics Engine. Disconnecting this functionality from the frontend allows for easy reusability of computed results and simplifies the API access for visualization and interaction components. D3.1 and D3.2 furthermore specify "Entity resolution" and "Entity ranking." Both of these are directly executed on the heterogeneous information network, rendering graph transformations a basic functionality of the Visual Analytics Engine. Furthermore, the results, after applying parameter changes, have to be compared to the cached results to allow per-session change representation in the Visual Explorer.

While the priorly mentioned deliverables are already completed or currently in progress, upcoming deliverables will afford further extensions of the Visual Analytics Engine to new analysis tasks. The forthcoming tasks "Link prediction" and "Community detection," as defined in D3.3, will also afford graph transformations and user-driven parameter refinement. Furthermore, per-session change indication will play a crucial role in these tasks.

Finally, deliverable D3.4, "Change management," will provide monitoring functionalities on the heterogeneous information network. This will involve higher-level graph abstractions as well as the persistent storage of different metrics over time.

1.3.2. Results Visualization & Parameter Tuning

All of the tasks elaborated above will support two main applications: results visualization and parameter tuning. As part of the visual analytics process, this describes the iterative cycle between model and results visualization, knowledge generation, and model refinement. Besides the functionalities mentioned explicitly, all of the work packages and tasks of SmartDataLake that are involved in any kind of visualization or interactive visual analytics will afford a corresponding mapping in the implementation of the Visual Analytics Engine and the Visual Explorer. This might range from pure interface abstractions up to complex data transformation workflows. Summarizing, the Visual Analytics Engine is the central interface between all automated and user-interfacing components, while the Visual Explorer provides user-interfaces, results and model visualizations, as well as interaction techniques.

2. Implementation of SDL-Vis

SDL-Vis is the visualization layer of SmartDataLake, enabling visual analytics on the heterogeneous data foundation and mining pipeline. This section describes the programming languages and libraries used to implement the Visual Explorer and the Visual Analytics Engine applications, i.e., the two main parts of SDL-Vis[2]. The tools were chosen to ensure reliability and reusability while providing modern interfaces. By implementing the backend in Python, we can make use of the rich set of libraries for scientific computing and data processing that are readily available. Implementing the frontend as a web application allows it to be easily accessible and provide state-of-the-art user interfaces.

2.1 Visual Explorer

Visual Explorer provides the frontend for SDL-Vis, and therefore also acts as the main user interface for SmartDataLake. This entails different stakeholders and groups setting requirements regarding the implementation of this interface. For instance, pilot partners may want to integrate SmartDataLake in their existing toolchain and be able to reuse parts of their existing workflow; research partners may want to integrate their customized views that were built during development; external users may need an easy way to set-up and reuse the functionality of SmartDataLake.

To fulfill such requirements, Visual Explorer is realized as a web application, ensuring accessibility, reusability, and customizability. According to the 2019 StackOverflow Developer Survey [2], JavaScript was the most commonly used programming language in 2019. Its widespread use, as well as the success of the Node.js[3] runtime environment, has resulted in an enormous ecosystem of available, ready-to-use packages. The packages are maintained and distributed by the Node package manager [4], enabling rapid prototyping as well as the easy integration of existing, web-based solutions. To ensure code quality and easy reusability, Visual Explorer does not rely on plain JavaScript, but instead uses a statically typed superset of JavaScript, called TypeScript[5].

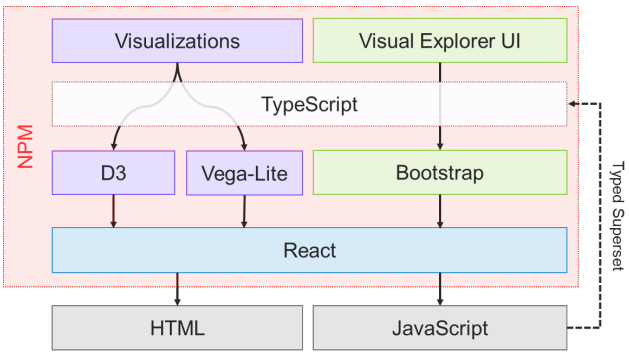

Figure 5: Technologies and libraries used for the implementation of Visual Explorer.

Furthermore, Visual Explorer uses the React[6] UI-library to build responsive, modular, and scalable user-interfaces. Bootstrap[7,8] brings in

[2] https://github.com/smartdatalake/sdl-vis
[3] https://nodejs.org/
[4] https://www.npmjs.com/
[5] https://www.typescriptlang.org/
[6] https://reactjs.org/
[7] https://getbootstrap.com/
[8] https://react-bootstrap.netlify.com/

pre-built, ready-to-use frontend-components (tabbed-view, buttons, sliders, etc.) and CSS styles that are up to date with current state-of-the-art user interfaces. Customized, non-standard visualizations are realized as custom React components, only using the convenience functions of D3[9] (d3-scale, d3-array, d3-path, etc.). This way, changes in the document object model[10] (DOM) are completely handled by React, ensuring a consistent state and the reliable update of all visualization components on data or viewport changes. For standard visualizations (line charts, bar charts, scatterplots, etc) or spatial and temporal data, Visual Explorer integrates the Vega-Lite[11] high-level grammar for interactive graphics. It allows to show visualizations for arbitrary attributes in the examined dataset without having to build customized solutions for each of them. Each attribute is annotated with tags (numerical, sequential, time-series, etc.) that indicate which type of visualization can be applied. Figure 5 shows an overview of the most important technologies and libraries used in Visual Explorer and how they relate to each other.

2.2 Visual Analytics Engine

The Visual Analytics engine is implemented in Python, which has gained relevance for data analysis, data processing, and machine learning applications in the last few years and, according to the 2019 StackOverflow Developer Survey [2], is the fastest-growing major language today. A large number of packages for data loading, processing, aggregation, and visualization makes Python an ideal choice for the implementation of the Visual Analytics Engine. Furthermore, with Flask[12], an easy solution for providing services via REST API is available. Similar to NPM in JavaScript, pip[13] is a package manager for Python, over which all of the used packages can directly be accessed. For the data processing steps, Pandas[14], SciPi[15] and Numpy[16] are used. For the graph-related tasks of SmartDataLake, such as the representation of heterogeneous information networks, NetworkX[17] provides both, appropriate data structures (node-link-representation, JSON[18] import/export, etc.), as well as basic graph algorithms (clustering, distance measures, traversal, etc.).

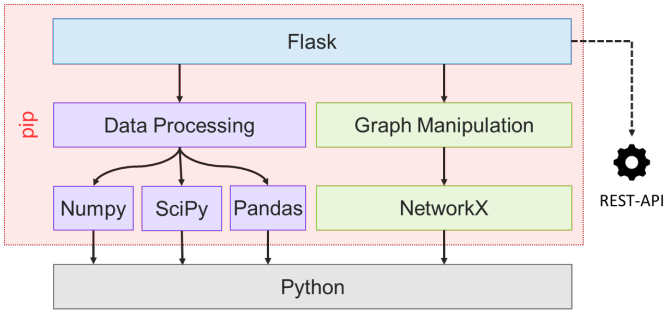

Figure 6: Tools and packages used for the implementation of the Visual Analytics Engine.

The REST API exposes the services of the Visual Analytics Engine to be accessed over the network. For each service, it implements a route that allows POST requests with a JSON payload to pass parameters.

[9] https://d3js.org/
[10] https://dom.spec.whatwg.org/
[11] https://vega.github.io/vega-lite/
[12] https://flask.palletsprojects.com/
[13] https://pypi.org/project/pip/
[14] https://pandas.pydata.org/
[15] https://www.scipy.org/scipylib/index.html
[16] https://numpy.org/
[17] https://networkx.github.io/
[18] https://www.json.org/

3. Visualization and Interaction Panels

Each of the analysis tasks in the SmartDataLake pipeline affords specific visualization and interaction designs. While the single components and functions do not share many mutual units of analysis, they still work together on the same data foundation, contributing to the overarching knowledge generation process. Understanding and refining the results of single individual components makes the complex pipeline more accessible and reduces the amount of information the user has to handle during one task.

The Visual Analytics Engine maps the required functionalities of each task to individual processing workflows, as described in the following subsections. Each task and workflow corresponds to a single visualization and interaction panel in the Visual Explorer user interface, as shown in Figure 7. For ongoing tasks, we present visual and textual descriptions of these workflows, showing the current state of the implementation. Upcoming tasks will be realized accordingly; however, at this point, discussions about the appropriate interfaces and functionalities are still in progress.

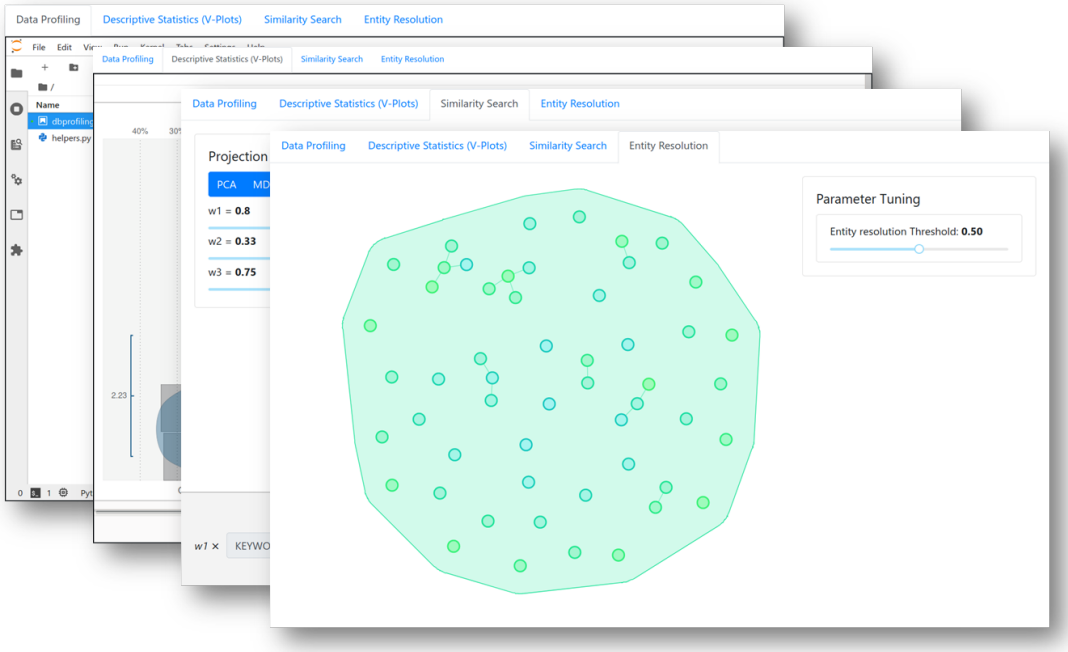

Figure 7: The user-interface of the Visual Explorer is divided into multiple *Visualization and Interaction Panels*, each of them targeting one specific analysis task in the SmartDataLake analysis pipeline.

The design of the user-interface of Visual Explorer with a detailed description of the distinct visualization and interaction panels is included in D4.1, "Interactive visual analytics model." It describes the tasks to be covered by the visual analytics layer as well as how human and automated parts of the data processing pipeline interact. The following sections focus on the inner workings of the Visual Analytics Engine to prepare data, models, and interfaces, as well as on how this data is then represented by the task-specific panels of the Visual Explorer.

3.1 Data Profiling

Data profiling is the most crucial analysis task regarding the storage layer of SmartDataLake. The quality of the analysis results is heavily influenced by the characteristics of the data sources themselves and the capabilities of the data collection and virtualization engine. Since data profiling is a highly workflow-driven task, dependent on data type, data quality, customer requests, and analysis goal, there is no all-in-one solution for the visualization layer. Instead, the analyst needs to feel his way forward slowly, further increasing the complexity of the analysis step-by-step while getting familiar with the dataset. Therefore, Visual Explorer includes a full Jupyter Lab environment with a Python kernel, enabling highly customized workflows and allowing to reuse existing solutions of our pilot partners.

As shown in Figure 9, the virtualization layer (Proteus and RAW, see D3.1, "Query engine over virtualized data") abstracts the heterogeneous data sources of SmartDataLake to be accessed by structured query language (SQL). Results are returned as relational data (tables with rows and columns), encoding data attributes and values. However, to analyze the rather complicated data foundation of large data lakes, additional information on data provenance and database schemata is crucial for the analysis. Therefore, besides querying the data values themselves, the data virtualization layer provides access to structural information of the data foundation.

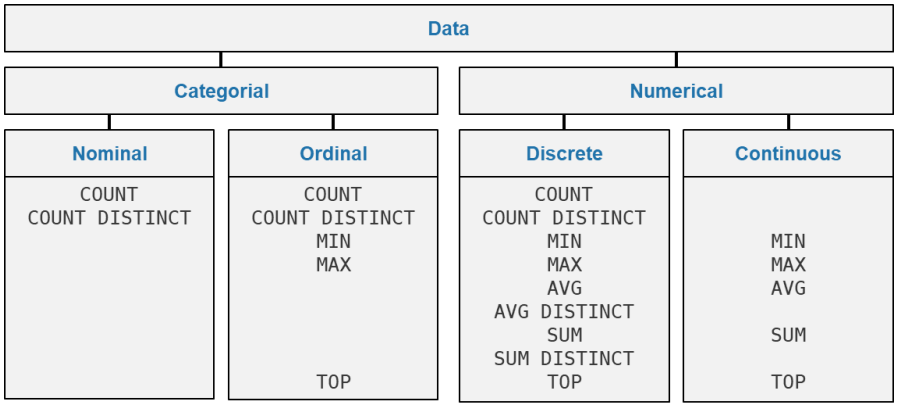

Figure 8: Types of operations that can be issued to the approximate query processing engine. For descriptive analytics, often approximate results are sufficient to reach the analysis goal, e.g., for histograms or record counts.

The vast datasets involved in SmartDataLake (e.g., Atoka Companies, 120 GB JSON) render exact storage access and data consolidation an expensive operation. However, for data profiling, approximate results are usually sufficient to reach the analysis goal. Therefore, with the approximate query engine (AQP, see D3.2, "Data synopses for approximate analytics"), SmartDataLake implements a component enabling the efficient analysis based on approximate results.

As Figure 9 depicts, accessing the AQP works analogously to querying the data virtualization layer by SQL. However, approximate results only make sense for aggregating queries, for example, SUM or COUNT operations. Figure 8 lists the types of operations that may result in a performance speedup when issued to the approximate query engine instead of querying exact results directly from the data virtualization layer. As also encoded in Figure 8, based on the data type of a table column, only a subset of aggregating operations can be applied. For example, it is not useful to SUM over categorical data. While connecting to the storage layer directly using a JDBC driver is possible and useful for advanced data profiling task, the Visual Analytics Engine also provides a default set of data aggregation and transformation techniques. This abstracted data access enables the targeted implementation of more complex models such as histograms or kernel density estimations, without having the user to implement complex SQL queries and workflows. The data is queried by using a high-level grammar similarly to Vega-Lite specifications [19] [3]. Besides pre-aggregated data, also schema information, as well as a set of projection techniques, are provided.

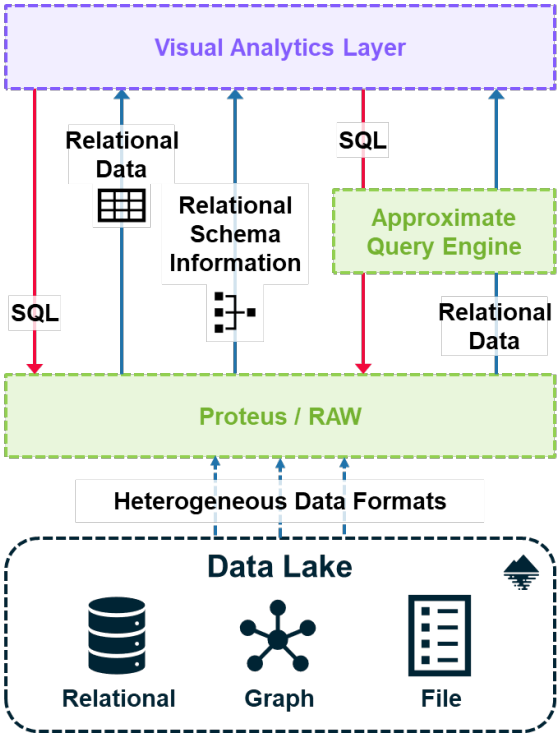

Figure 9: Storage access formats.

3.1.1 (V-Plot) Descriptive Statistics

Descriptive statistics is an essential part of the data profiling task. By analyzing and exploring numerical data and its distributions, useful insights can be gained for further analysis. For example, by examining the frequency density of a distribution (e.g., by using a histogram), the analyst can assess the overall shape of the data, identify possible outliers, evaluate the uncertainty of the data points, or estimate replacements for missing values.

With V-Plot [4], we present a component specifically tailored to the visual analysis of data distributions. Based on a comprehensive user study, a variety of chart types was evaluated according to their suitability for different analysis tasks. From the study results, we derived a workflow that supports the user in selecting useful chart types to solve a specific analysis task. An interactive wizard guides the user in specifying his analysis task according to common categories, proposing a ranked list of charts suitable to solve these tasks. In addition to the recommended single charts, a hybrid chart is created, which combines all tasks in a single plot, called the "V-Plot," as shown in Figure 10.

[19] https://vega.github.io/vega-lite/docs/spec.html

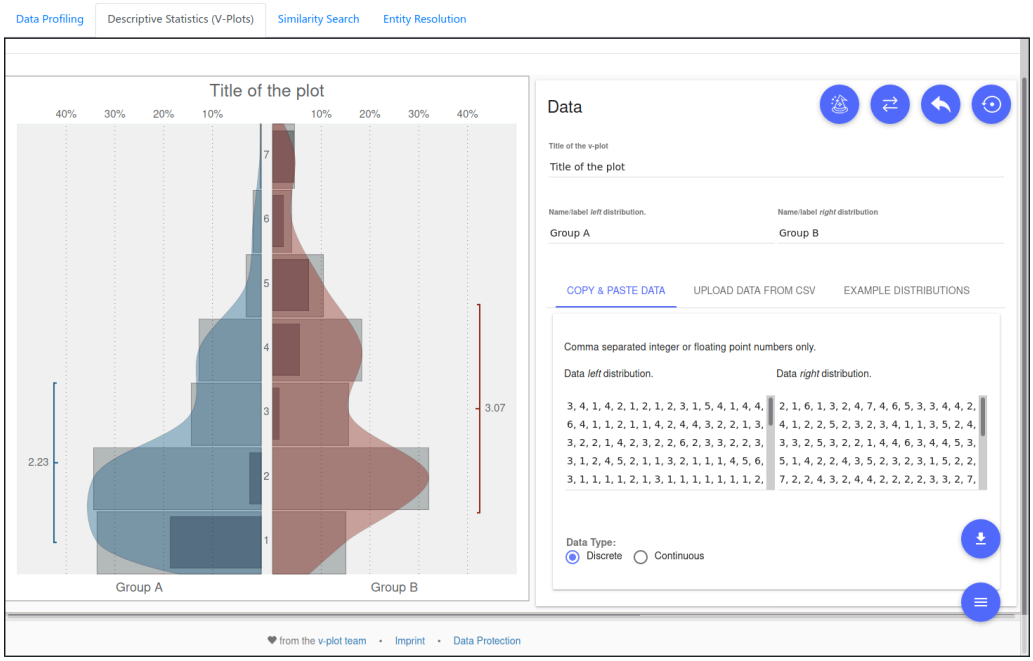

Figure 10: The descriptive statistics panel of the Visual Explorer. It helps in identifying chart types which supports the specified analysis tasks and allows to automatically generate a hybrid chart combining the tasks in one plot.

Through its a semi-automated approach, V-Plot provides fast insights on the numerical dimensions of a dataset, enabling the analyst to reveal relevant information in the distributions of single or multiple data columns. It, therefore, acts as an entry point to a more detailed analysis, helping the analyst to explore the data and identify further analysis questions.

V-Plot is designed to work on tabular data. More specifically, it allows choosing either one or two numerical dimensions of the dataset to be represented in the generated charts. Therefore, to enable data selection and access, the Visual Analytics Engine has to support two main functionalities:

1. Provide information on available tables and columns in the data foundation
2. Query data from the selected table

Figure 11 shows how the described functionalities are implemented and how data is exchanged and transformed between the components. The V-Plot component queries the available tables together with the respective attributes and data types using the REST API. The Visual Explorer Engines issues a query to the data virtualization layer, triggering a response containing schema information. The schema information is transformed from a tabular format to JSON and returned in the response. Now the analyst can select the desired tables and columns, which are then queried from the Visual Analytics Engine REST API, containing the relevant information as JSON payload. The request is translated into SQL queries, which are then executed on the data virtualization layer. Returning tabular data, the Visual Analytics Engine stringifies the values as CSV and returns it via POST response.

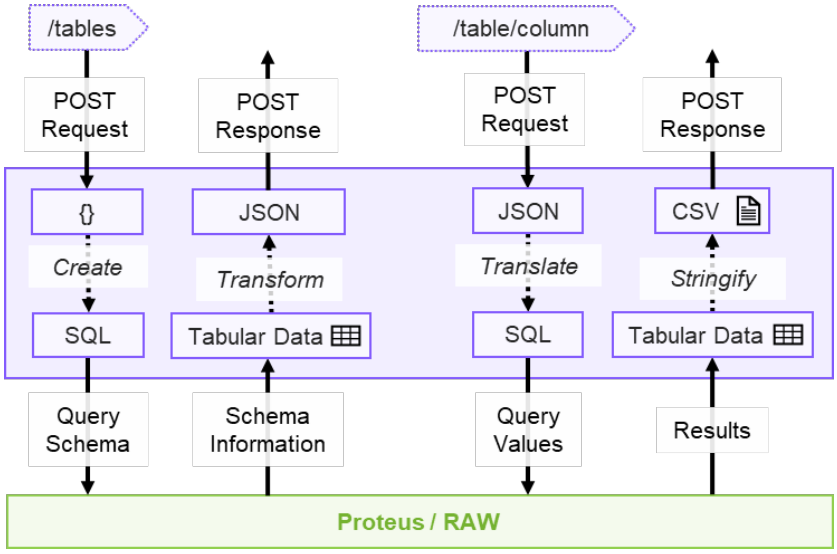

Figure 11: The functionalities that the Visual Analytics Engine has to support for the descriptive statistics (V-Plot) panel of Visual Explorer.

3.1.2 Jupyter Lab

From the descriptive statistics panel (V-Plot), the analyst might have generated new hypotheses or derived new analysis questions that have to be answered with more complex models and visualizations. Therefore, for advanced data profiling tasks, we include a Jupyter Lab [5] environment into Visual Explorer, allowing the implementation of highly customized workflows for data profiling. By building on Python libraries, we can provide a rich set of visualization- and data mining packages, such as SciPy, Numpy, and Pandas. Figure 12 shows a screenshot of the data profiling panel.

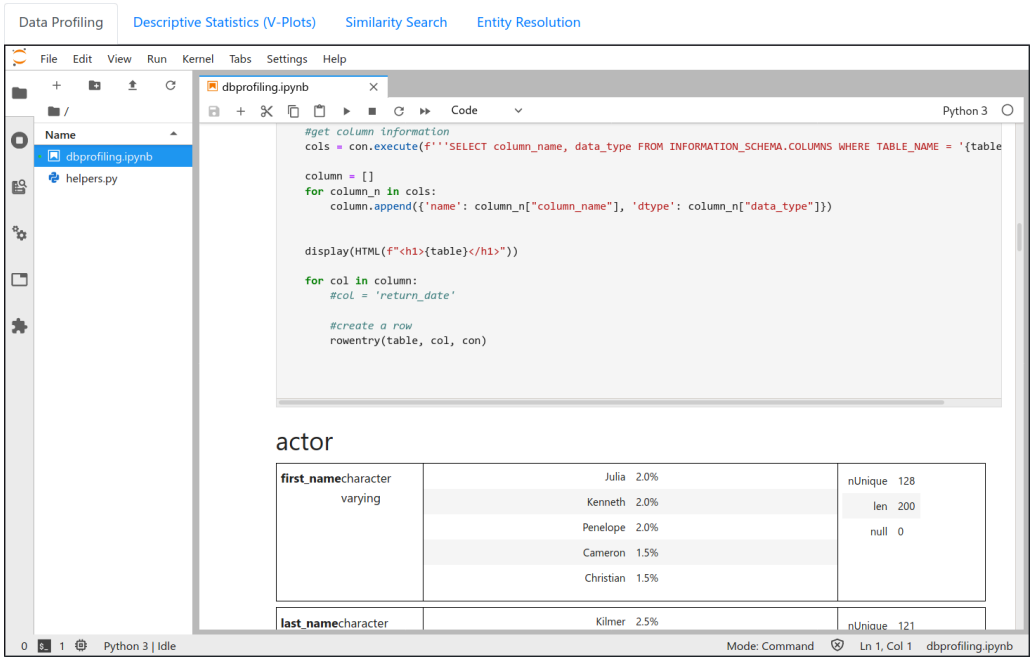

Figure 12: The data profiling panel of Visual Explorer, providing a Jupyter Lab environment. The Python kernel with a large set of pre-installed packages for scientific computing, data visualization and data mining packages allows for highly customized data profiling workflows.

The Visual Analytics Engine has to implement the following three main functionalities to support data profiling from Jupyter notebooks, shown in Figure 13:

1. Provide information on the structure of the data foundation (schema)
2. Provide a default set of statistical descriptors for data values
3. Cache the computed results

As an entry-point to the data profiling process, the analyst has to understand the structure of the underlying data. Therefore, analogously to the descriptive statistics panel, schema information can be queried by the REST API. For pre-aggregated data, an additional API endpoint allows specifying data transformations that should be applied to the tabular data returned from the data virtualization layer. This results in statistical descriptors of varying formats, for example, a dictionary encoding a histogram of the data. The transformed data is cached by the Visual Analytics Engine for later reuse. Finally, the results are stringified to the JSON format and returned in the POST response.

While querying the endpoint for pre-aggregated data, an additional precision parameter can be specified, indicating that the Visual Analytics Engine is allowed to query the results from the approximate query engine, resulting in a potential speedup of the query process.

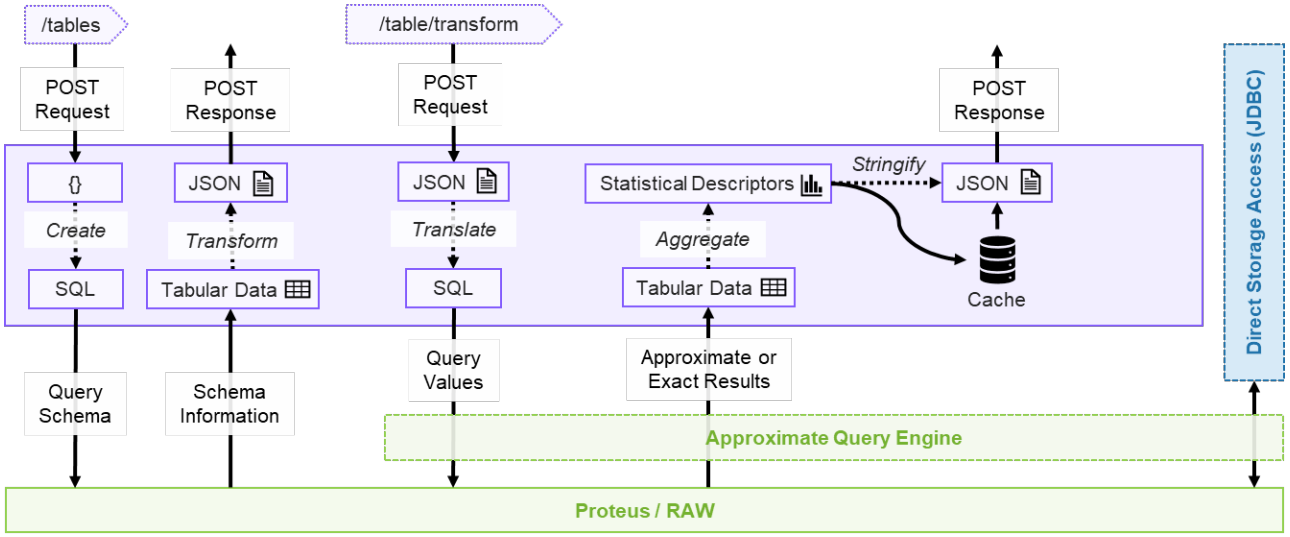

Figure 13: The functionalities that the Visual Analytics Engine has to support for the data profiling (Jupyter Lab) panel of Visual Explorer.

For more advanced data profiling applications, where the analyst needs access to the raw results from the data virtualization layer, the Visual Analytics Engine backend can be omitted. In this case, a JDBC connection between the Python application layer (Jupyter Lab Panel of Visual Explorer) and either the approximate query engine or Proteus / RAW allows customized queries. For example, this could be useful for iterative data access or complex queries that cannot be abstracted in the JSON transform specification.

3.2 Similarity Search

Search is the most powerful utility to efficiently focus on a relevant subset of a large data collection. However, since the knowledge about the entities of interest is limited beforehand, the optimal search parameters are typically not known. Furthermore, for specific use-cases, finding a group of the most similar results might be of interest. For example, to rate the value of a new company, similar company profiles in geographic proximity can provide useful starting points. Therefore, identifying entities that are close to the desired search parameters is essential to explore large datasets and focus on regions of interest. The similarity search panel offers this functionality on the heterogeneous data foundation of SmartDataLake.

Figure 14: The similarity search panel of the Visual Explorer. Queries can be specified in the query builder. The results are projected, grouping entities that share more common attributes. The parameter tuning section allows to change the weights of each attribute. Possible outcomes of parameter changes are already indicated in red (decrease) and green (increase) when hovering a slider, allowing real-time analysis on time consuming operations.

As shown in Figure 14, the query builder allows specifying the root query for the search. Since distance measures between different data types or domains can not be compared directly, additional weight parameters enable the analyst to balance the influence of the various search attributes according to his understanding of domain and analysis goal. However, choosing a good set of weights is difficult, since their abstract influence on the results of the similarity search engine is hard to grasp. Therefore, the similarity search panel includes a parameter tuning section, allowing to iteratively refine the weight parameters, enabling the analyst to approach the ideal solution gradually. To preserve the real-time capabilities of the Visual Explorer, we provide speculative execution on different combinations of weight parameters. By indicating possible outcomes before actually changing a parameter, the analyst can estimate if the change would benefit the analysis process. By hovering a weight slider in the parameter tuning section, the visualization is updated, indicating the result sets for both positive and negative weight changes.

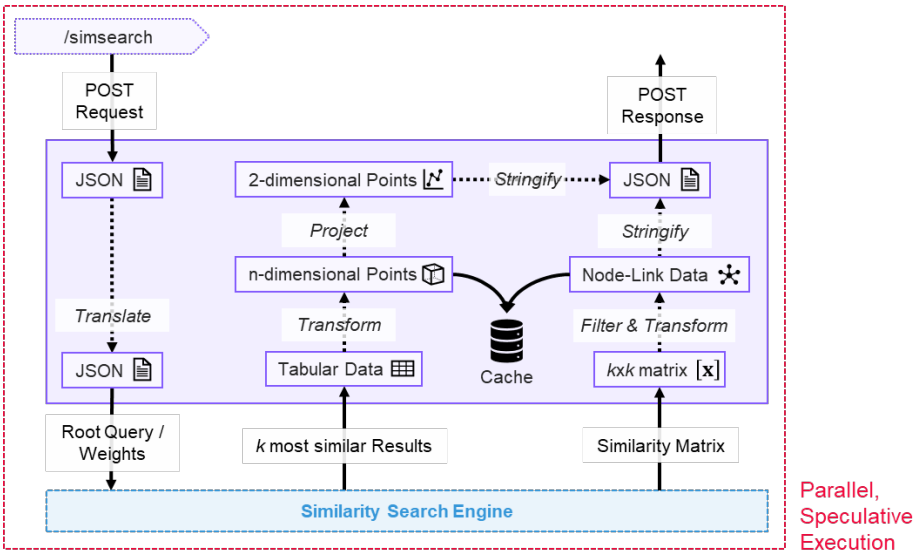

Figure 15: The functionalities that the Visual Analytics Engine has to support for the similarity search panel of Visual Explorer.

The Visual Analytics engine implements the following main functionalities to support the similarity search panel, detailed depicted in Figure 15:

1. Query the similarity search engine with a set of attributes and a set of weights
2. Transform the returned *k* most similar results to n-dimensional points
    a. Project the n-dimensional points to 2D using a user-specified dimensionality reduction method
3. Filter and convert the returned *kxk* cross-similarity matrix to a node-link representation
4. Cache the results for future queries

After using the query builder to create a root query and setting the initial weights, the similarity search panel sends a POST request to the REST API of the VIsual Analytics Engine. The engine translates the JSON payload and issues a query to the similarity search engine. As a result, the top-*k* similar entities, as well as a cross-similarity matrix, are returned. The entity table is transformed into an n-dimensional point cloud, which is then projected using the specified dimensionality reduction technique (e.g., UMAP, PCA, MDS). To avoid visual clutter and reduce the amount of submitted data, the cross-similarity matrix is filtered for the top-*n* similarity scores and transformed into a node-link representation. All the computed results for the current set of attributes and weight parameters are cached for future access. By executing this process speculatively for the local neighborhood of each weight, possible future parameter combinations can already be executed and stored, enabling real-time parameter refinement in the frontend.

3.3 Entity Resolution

Meaningfully merging multiple, heterogeneous data sources is a complicated task. Different data entities referring to the same real-world entity are often hard to detect since they might be described by a different set of attributes. For example, the Wikipedia page of a company and the phone book entry of a company might refer to the same real-world entity while not sharing any common attribute. Automatically resolving such trivial connections in the heterogeneous information network of SmartDataLake is an essential pre-processing step to be able to detect other, more complex relationships during the analysis.

Entity resolution, as described in D3.1, "Similarity search, entity resolution, and ranking," tackles this problem by transforming the heterogeneous information network according to pre-defined rules, so-called graph generating dependencies (GGDs). By defining a source graph pattern and a target graph pattern, GGDs encode how the heterogenous information network has to be transformed to fulfill the GGD. This allows prescribing patterns that insert *sameAs* edges in the network, indicating that two or more data entities refer to the same real-world entity. To transform the HIN according to the analyst's knowledge, the GGDs have to be iteratively refined. By visually observing the updates in the graph, the analyst can assess the quality of his changes and further improve the parameter choice.

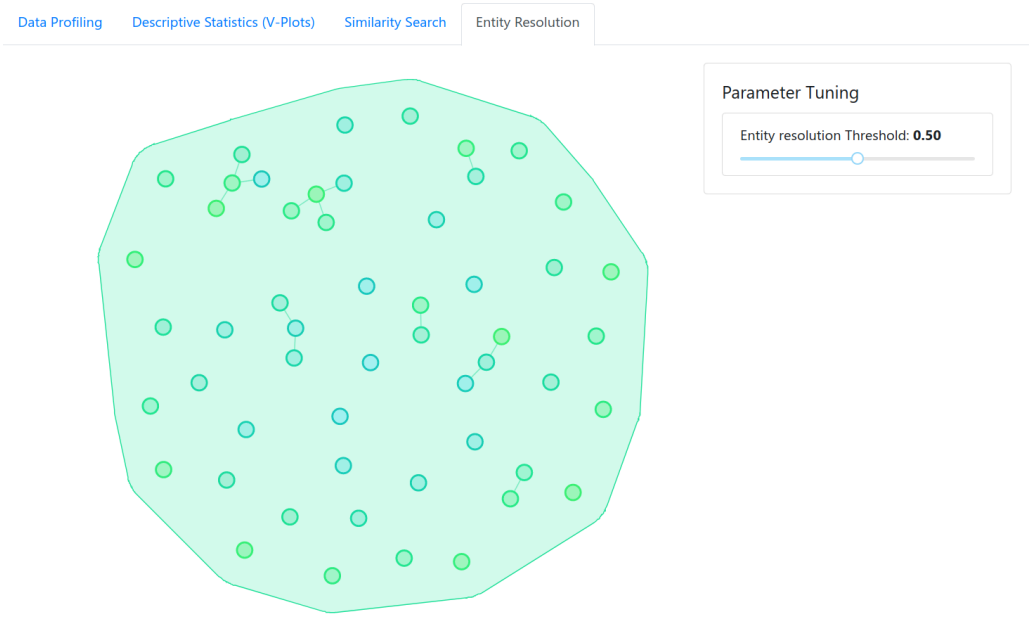

Figure 16: The entity resultion panel of the Visual Explorer, showing the nodes of a local group in the heterogeneous information network together with the edges inferred by resolving graph generating dependencies (GGDs). The panel is currently under development; an interface to interactively refine GGDs, change indication, as well as direct interactions with the edges and nodes of the graph will be added in the course of the project.

Figure 16 shows the entity resolution panel of the Visual Explorer in the current development state, showing a local group of the HIN graph together with the links inferred by resolving a GGD. In the course of the project, the following features will be added:

- An interface to interactively refine the GGDs
- Visual change indication in the graph after the GGDs have been updated
- Navigation methods over the HIN (zooming, panning, selecting regions of interest)
- User interactions to refine the GGDs by directly interacting edges and nodes of the HIN graph

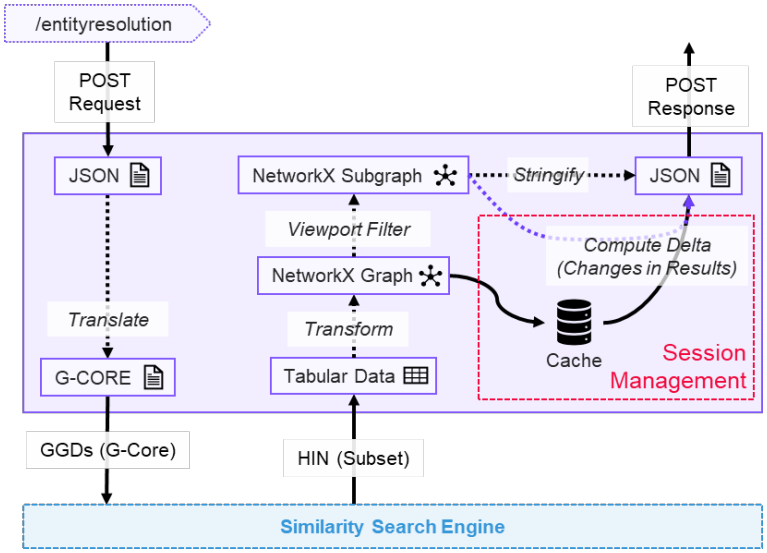

Figure 17: The functionalities that the Visual Analytics Engine has to support for the entity resolution panel of the Visual Explorer.

The Visual Analytics engine has to implement the following main functionalities to support the entity resolution task:

1. Query the entity resolution engine with the given GGDs
2. Transform the resulting graph in tabular form to a NetworkX graph representation
    a. Filter the graph according to the selected viewport in the Visual Explorer
3. Cache the computed results
    a. Use the previously cached version to communicate changes in the graph to the frontend (affording session management)

After specifying the GGDs that should be used to transform the graph, the entity resolution panel queries the Visual Analytics API endpoint for entity resolution. The Visual Analytics engine translates the JSON representation of the GGDs to a G-Core query that can be issued to the similarity search engine. The entity resolution engine transforms a subset of the HIN, since

executing the GGD on the full network takes a significant amount of time. Therefore, to preserve real-time analysis, the refinement of the GGDs is performed on a representative cluster of the network. When the user is satisfied with the chosen parameters, the GGDs can be applied to the full graph asynchronously. After transforming the HIN subset according to the specified GGDs, the similarity search engine returns a graph in the form of tabular data, which then is transformed into a NetworkX object. This allows using the graph algorithms of NetworkX, such as filtering and serialization. From the current viewport of the visualization frontend, the graph is further filtered to the visualized subset of links and nodes. Finally, the computed results are cached and compared to the latest stored version in the current user session. Changes in the graph get marked, and the resulting annotated graph is serialized and returned to the frontend.

3.4 Visual Exploration of Time Series Data

Since time series data makes up a substantial part of the heterogeneous data sources involved in SmartDataLake, developing visualizations and techniques to make them accessible is a fundamental requirement in the project. Therefore, in the following, we present an application for the targeted analysis of time series data, including algorithms and visualizations to detect and represent regions of interest. Currently, time series exploration and analysis is performed in a standalone application. Integration of the Time Series Explorer into the Visual Explorer is planned for future versions of SDL-Vis.

Time series data can provide significantly useful information for a variety of activities, from scientific research to industrial-level applications. Time series analytics and exploration are standard, yet demanding tasks that are usually performed by experts. However, obtaining useful insights from time series data can be significantly facilitated and be made more accessible to the average user via proper visualizations. Visualizing a time series just by plotting it in a diagram is the most straightforward way for obtaining fundamental insights (e.g., local/global trends, minimums, and maximums). Being able to interact with the plot, such as zooming in and out or selecting only subsequences of the time series to obtain relevant statistics quickly, can provide even more insightful information.

Our SDL-Vis time series exploration component contains an interactive application that enables the user to explore a time series dataset. For example, the application enables the discovery of *motifs*, i.e., patterns within a time series that are frequently repeated. The application offers the ability to visualize selected time series and perform several transformations on them, such as smoothing and z-normalization. The user can interact with the plotted time series, by zooming and panning to see specific parts of the plot. Furthermore, the application utilizes the similarity search over time series API from the HIN mining engine, to perform self-join on the data (see Deliverables D3.1, Section 3.4.2 and D3.2, Section 3.3). The resulting pairs from self-join, along with the corresponding local similarity intervals, are plotted, and the user can interactively explore them and obtain several useful statistics. Another feature that the interactive application provides is the calculation of the Matrix Profile [6] of a selected time series, which is a vector of Euclidean distances between each subsequence of a time series (considering a fixed window) and its nearest

subsequence. Based on the Matrix Profile, the application provides a visualization that plots the top 10 detected motifs, i.e., pairs of subsequences that are very similar to each other.

This component has already been tested with SPRING's time series stock data. Specifically, we have tested it on SPRING's historical data, containing daily stock closing values. The input data must be time-aligned to apply a self-join operation. Thus, we used the stock dataset described in Deliverable D3.2, Section 3.3.4.1. Each file of the dataset contains the daily (only for working days, 261 in total per year) closing values of a specific stock for the years 2007-2012. In total, the dataset contains 12,232 time-aligned stock time series of length 261 (working days) x 6 (years) = 1,566.

In the following, we present the time series explorer application (operating on SPRING's data), its functionalities, and the way users can interact with it to obtain insightful information regarding the various stocks. The application is developed in Python 3.7, using Jupyter notebooks with appmode[20] extension to generate the graphical environment and the Bokeh[21] library to generate the various interactive visualizations. The main view of the application is depicted in Figure 18.

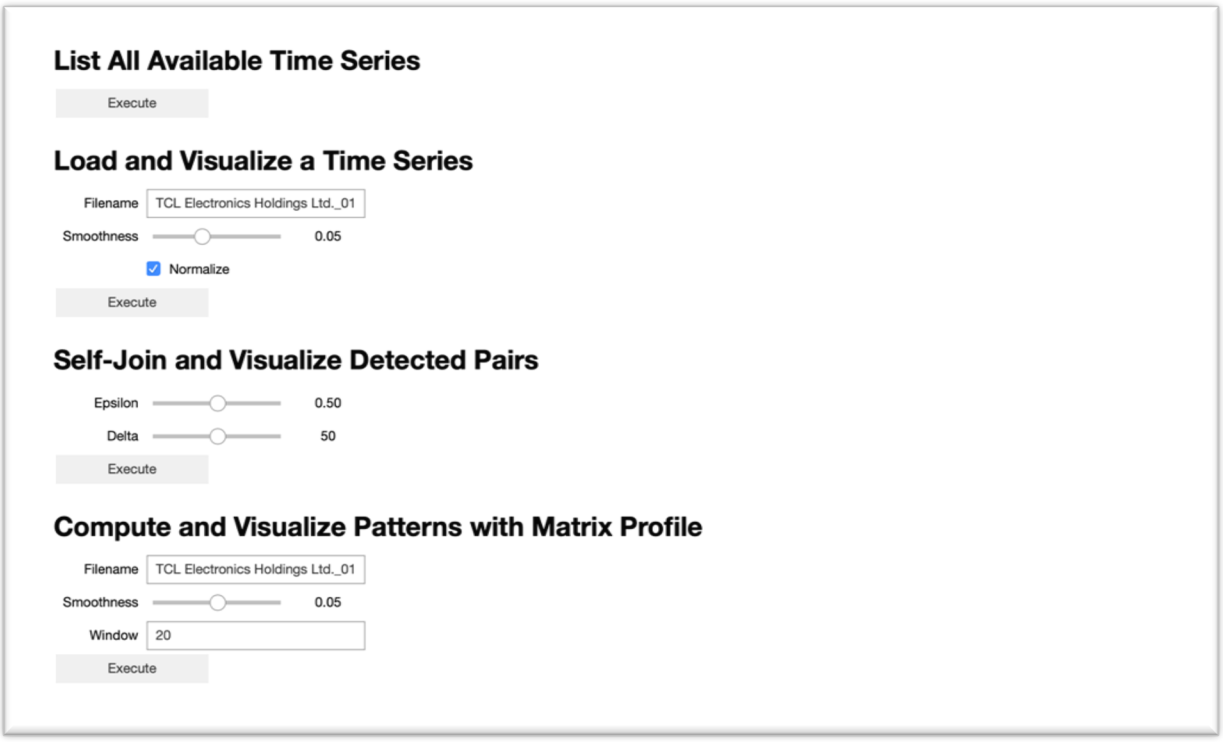

Figure 18: Time series explorer application's main view. The four main functionalities are accompanied by sliders, fields and checkboxes, corresponding to function arguments and enabling an interactive parameter exploration.

Each functionality has an "Execute" button that triggers it to avoid unnecessary updates. The various sliders, fields, and checkboxes are used to set the values of certain variables that are required for the execution. We explain these functionalities in more detail below.

[20] https://github.com/oschuett/appmode

[21] https://bokeh.org

1. List All Available Time Series

The most straightforward functionality offered by the application is to list all available time series. It directly outputs the filenames of all stock time series files found in the dataset's folder. Figure 19 depicts the result after pressing 'Execute.'

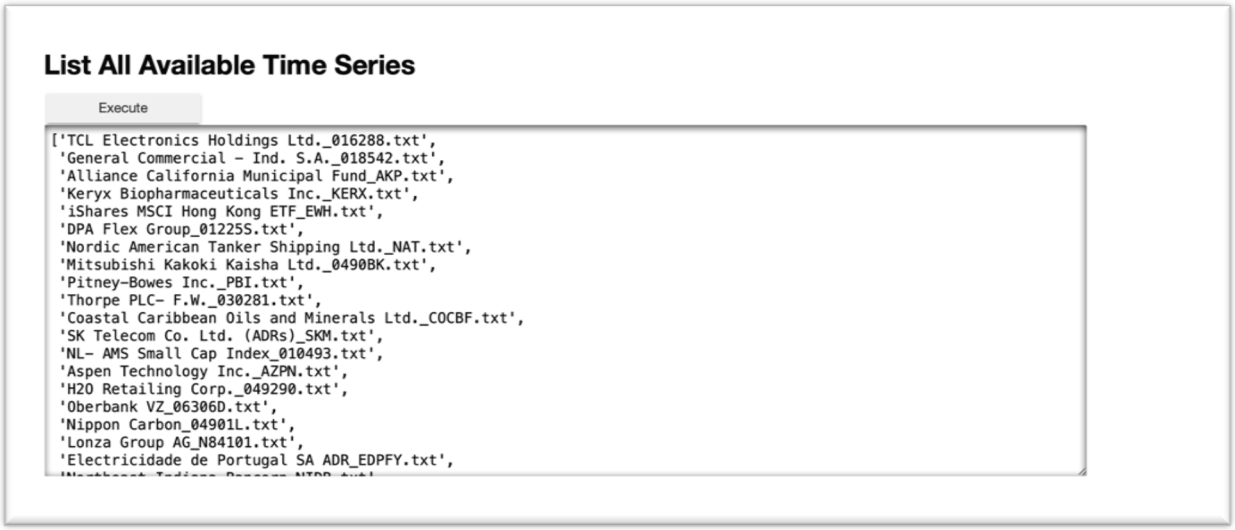

Figure 19: List view of all the available stock time series.

2. Load and Visualize a Time Series

This functionality generates an interactive diagram of a selected time series. Figure 20 depicts the generated visualization for a selected time series from SPRING's stock data.

As shown in the figure, in the field 'Stock,' the user can provide the filename of the time series to be plotted. By clicking on the 'Execute' button, the application plots a raw time series (black line), along with its smoothed version (red dashed line). The slider named 'Smoothness' determines the smoothing factor value. For smoothing, we use the LOESS algorithm, which stands for Locally Estimated Scatterplot Smoothing [7]. For each time series value, LOESS uses its window nearest neighbors on the x-axis -weighted according to their distance to the current value- to fit a polynomial regression model of a given degree. This model is then used to determine the value at each timestamp. The higher the window value, the smoother the results. In our case, the smoothing factor determines the size of the window, since it represents the fraction of the data to be used when estimating each value.

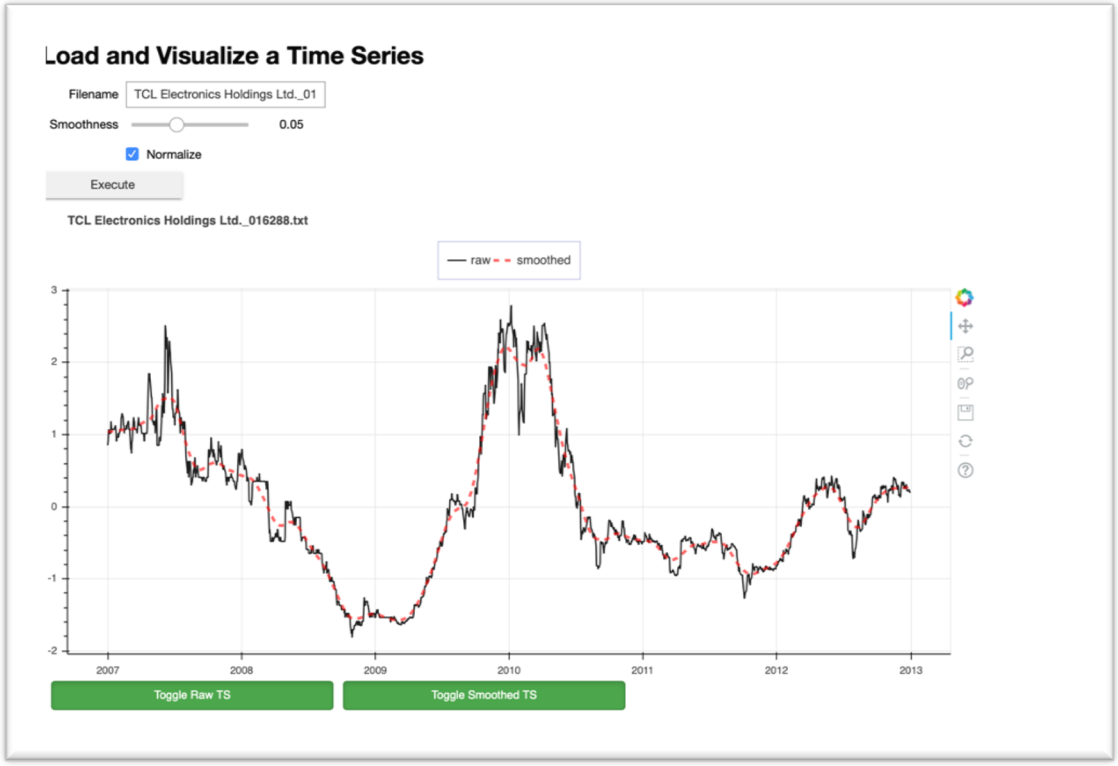

Figure 20: Loading and visualizing a time series (black solid line).The red dashed line shows its smoothed version using the LOESS algorithm.

The user can also pre-select whether the time series should be z-normalized before being plotted, by clicking on the 'Normalize' checkbox. Z-normalization ensures that the mean of the output time series is approximately zero, while the standard deviation is in a range close to 1. It is calculated using the below formula:

$$x_i' = \frac{x_i - \mu}{\sigma} \qquad i \in \mathbb{N}$$

where μ is the mean value of the time series and σ the standard deviation. By clicking on the 'Toggle Raw TS' and 'Toggle Smoothed TS' buttons at the bottom, the user can show or remove from the plot the raw and smoothed time series, respectively, in case she only wants to explore one of them. Finally, using the standard tools provided by the Bokeh library (on the right part of Figure 20), the user can move around the view, zoom in or out, save and refresh the plot. For example, Figure 21 shows a part of the smoothed time series, after zooming in.

Figure 21: Smoothed time series after zooming in. The user can toggle the smoothed or raw version of the time series using the buttons on the bottom.

3. Run Self-Join and Visualize Selected Pairs

This functionality uses the time series API described in Deliverable D3.2, Section 3.3 in order to perform self-join on a time-aligned time series collection. The resulting pairs are plotted in an interactive visualization, while offering several statistics regarding the members of each pair. Figure 22 depicts an example of the generated visualization, for a selected pair of the self-join results. The time series are z-normalized in a pre-processing step to eliminate amplitude discrepancies among time series and focus on structural similarity.

On the top part of the visualization, using the 'Epsilon' and 'Delta' sliders, the user can pre-select the ε and δ thresholds. Specifically, ε is the maximum per timestamp value difference to detect a local similarity, and δ is the minimum number of consecutive timestamps where ε is satisfied, to consider two time-series as locally similar. In our example, δ is measured in the number of days. Regarding ε, since the data are z-normalized, it corresponds to standard deviations from the mean value. When 'Execute' is clicked, the system calls the proper API function for self-join, and retrieves the resulting pairs. Then, the visualization is rendered.

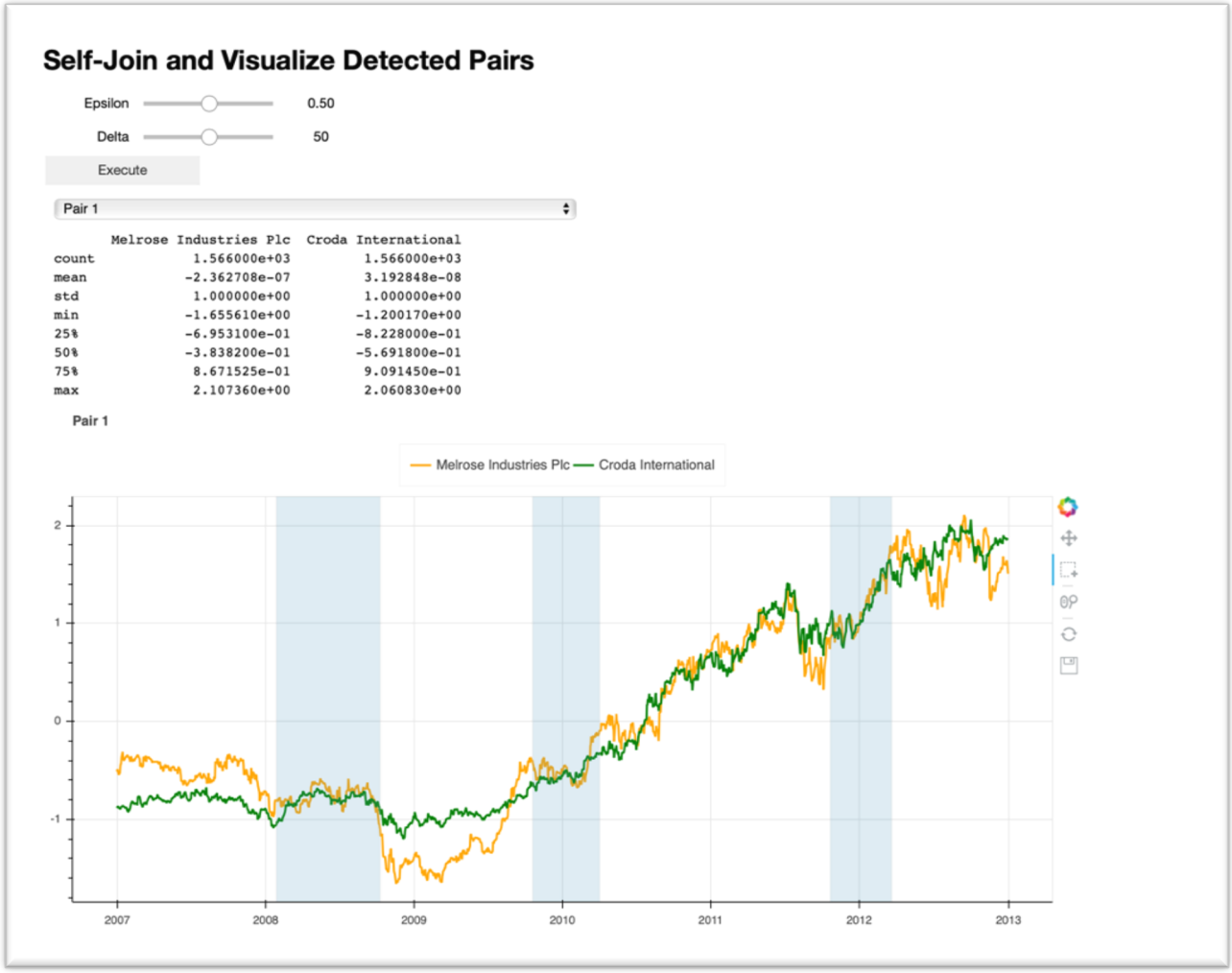

Figure 22: Self-Join and Visualize Detected Pairs. The blue-shaded rectangles are the time intervals where the two time series are locally similar.

Using the drop-down list depicted in Figure 22 under the 'Execute' button, the user can select another detected pair to visualize. Below this list, there are several statistics for each time series of the pair (in the example, we have Melrose Industries Plc and Croda International). Specifically, the following statistics are available:

- **Total count (count)**: The total number of values in the time series.
- **Mean value (mean)**: The mean value of the time series values.
- **Standard deviation (std):** The standard deviation of the time series values.
- **Minimum value (min):** The minimum value encountered within the time series.
- **Percentiles (25%, 50%, 75%):** The 25%, 50% and 75% percentiles of the time series values, i.e., the percentage of observations that fall below the shown value.
- **Maximum value (max):** The maximum value encountered within the time series.

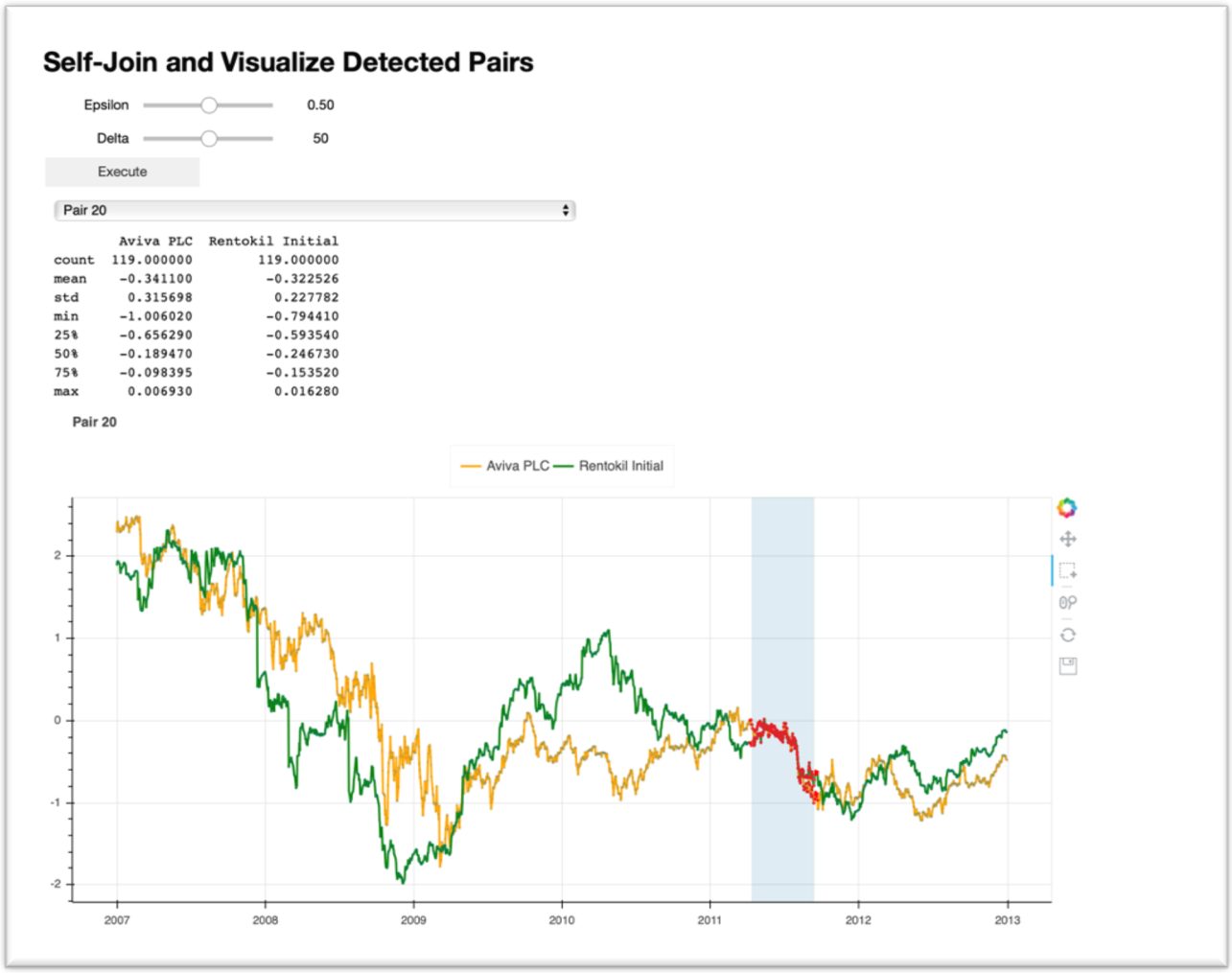

Figure 23: Selection of a specific interval. The selected interval is annotated with red color. The statistics pane above the plot is updated accordingly.

The generated interactive plot, which depicts the members of the selected pair, is located on the bottom part of the visualization. As previously, we have all the necessary functionality provided by Bokeh, such as zoom in and out, refresh and save the figure. Each pair member time series is plotted using different colors (green and orange). The blue shaded areas indicate the time intervals where the two time-series are locally similar. In the example depicted in Figure 22, we have three such intervals, in different parts of the time series. Another useful feature in this visualization is the ability to select a specific time interval and have the above statistics updated for the subsequences within the selected interval. Figure 23 depicts such a case. Note that, in this example, a different resulting pair is selected (Pair 20). The interval where the two time-series are locally similar is manually selected and annotated with red color. Finally, the statistics pane is automatically updated using the data in the selected interval (e.g., the timestamp counter is changed to 119).

4. Compute and Visualize Patterns with Matrix Profile

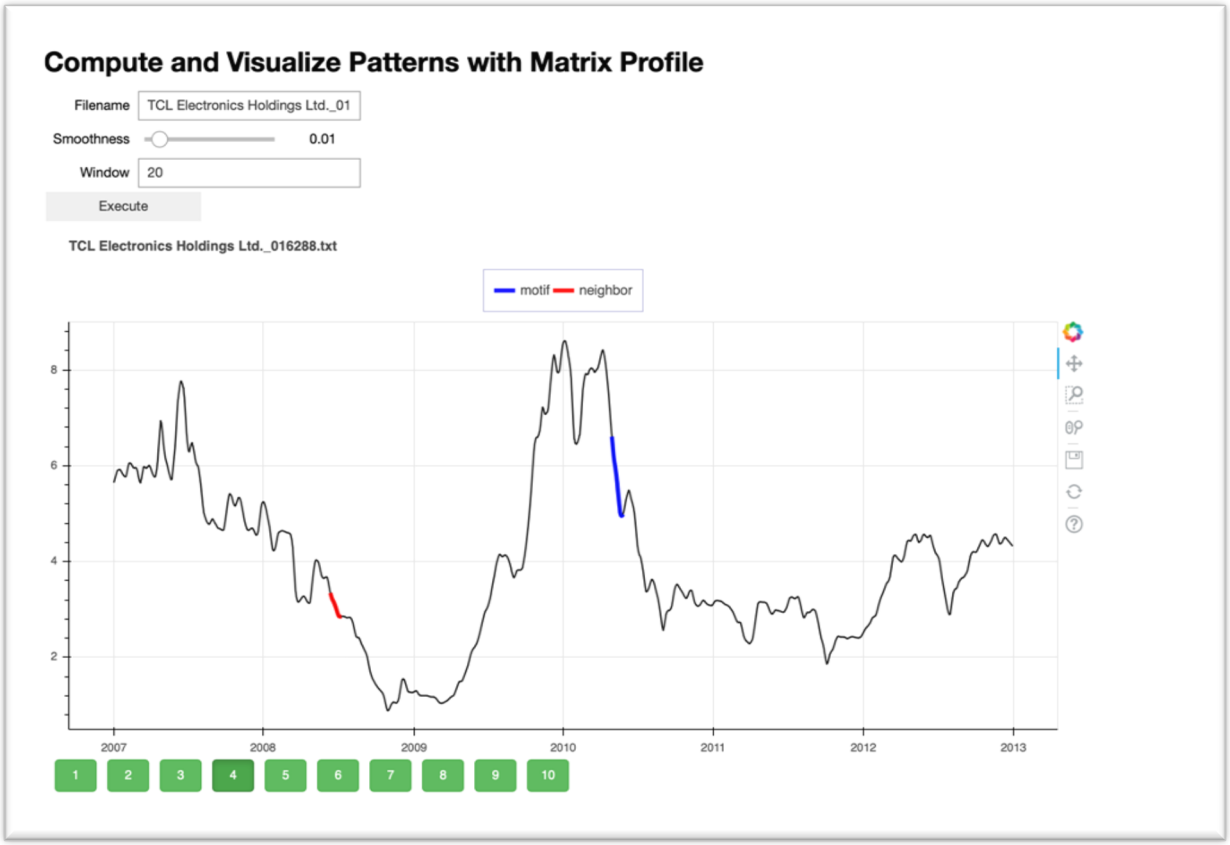

Figure 24: Identifying and visualizing top-10 motifs based on Matrix Profile. The motif is annotated in blue, while its nearest neighbor is highlighted in red.

Considering a pre-defined sliding window of length w_0, we can obtain all subsequences of a given time series having a length w_0. A widely researched operation is the all-pairs-similarity-join of the subsequences of a time series. This operation detects, for each subsequence, its nearest neighbor in terms of a given distance measure (Euclidean distance in our case). The Matrix Profile of a time series is a vector of the Euclidean distances of all the nearest neighbors for each subsequence [6]; it can be proven useful in various applications. For example, via the Matrix Profile, one can quickly obtain subsequence motifs of time series (i.e., pairs of similar subsequences) and discords (subsequences that differ significantly from all the rest): The smallest distances in the Matrix Profile correspond to motifs. In contrast, the longest distances correspond to discords. Discords and motifs can be thought of as patterns that are either unique or more frequently repeated within a time series, respectively.

Figure 24 depicts a running example of the Matrix Profile-based visualization. As in the simple time series visualization described in Section 0, the 'Filename' field is used to select the time series for which the Matrix Profile will be computed, and the 'Smoothness' selector applies optional smoothing to the time series, before the calculation. In the 'Window' field, the user can input the desired sliding window length, which will provide the time series subsequences. By pressing

'Execute,' the Matrix Profile is calculated, and the top-10 motifs are retrieved, by getting the 10 closest detected Euclidean distances, along with the corresponding subsequence positions on the time series. The user can press on any of the buttons on the bottom (annotated with numbers from 1 to 10) to select the corresponding motif. The selected motif, along with its nearest neighbor, is then plotted. In Figure 24, the 4th top motif has been selected, and it is drawn in the diagram in blue color. Its nearest neighbor is annotated with red. The user can select multiple motifs (and corresponding neighbors) to be depicted by clicking on more than one button on the bottom (Figure 24). As in the previous visualizations, all the basic functionality provided by the Bokeh library (zoom-in/out, move around, refresh, save) is also available here.

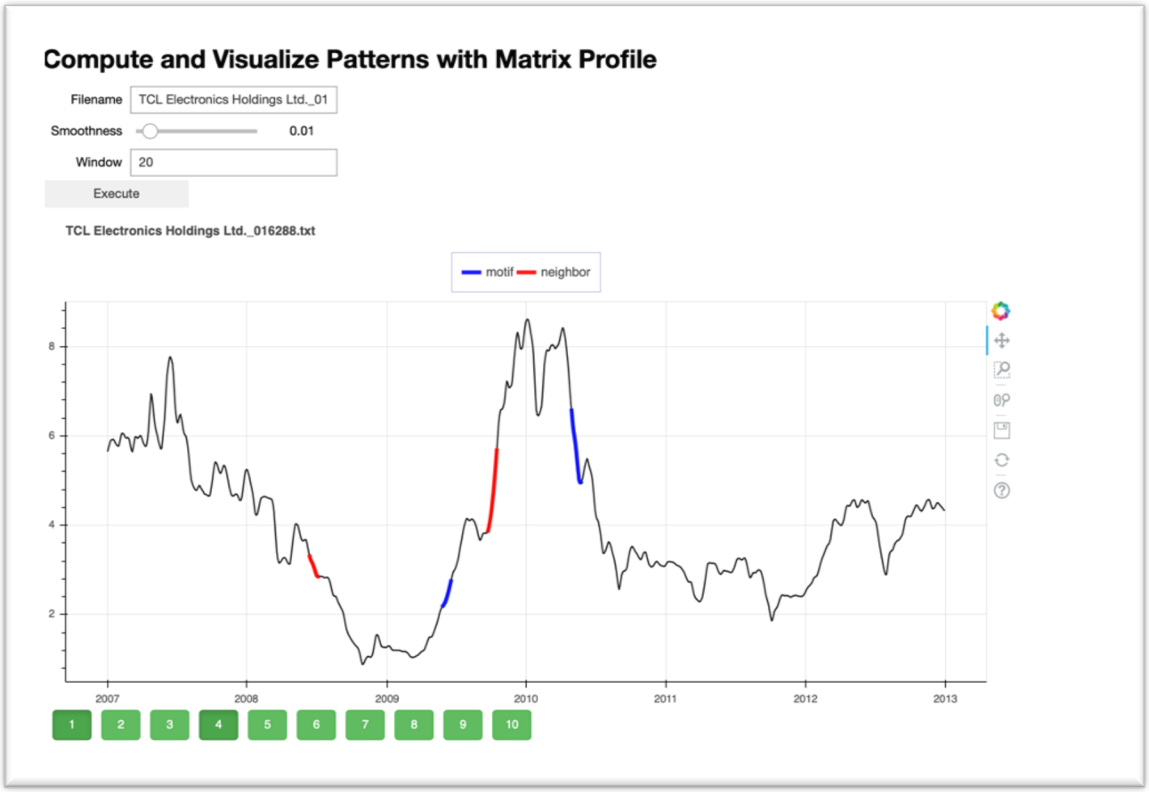

Figure 25: Visualizing multiple motifs simultaneously. Up to ten motifs can be selected and visualized simultaneously, using the buttons below the plot.

3.5 Visual Exploration of Top-k Spatial Regions

As described in Deliverable 3.1, "Similarity search, entity resolution and ranking," the spatial dimension plays a crucial role in many application areas (e.g., in geo-marketing, logistics, or urban planning). According to Tobler's first law of geography, "everything is related to everything else, but near things are more related than distant things" [8]. This becomes even more critical in the context of data lakes, which typically end up containing rich yet disparate spatial information from different sources, which, when put together, can generate value for the company (e.g., public repositories describing industrial activities at each location, news articles containing companies and their location, proprietary data about the customers of a company and their addresses). In SmartDataLake, we study the problem of top-k Best Region Search (BRS), which refers to ranking spatial regions based on their contents. This can be used, for example, to find those areas in a given city or country that contain the most significant number of companies of a particular type or generate the most considerable amount of revenue.

Given a set of points, the top-*k* Best Region Search problem (BRS) finds the optimal location of *k* rectangles of a specified size such that the value of a user-defined scoring function over its enclosed points is maximized. The user can consider the overlap between the set of selected top-*k* rectangles or detect non-overlapping regions. The top-*k* BRS problem has numerous applications related to location planning, e.g., a company that is searching for the optimal locations to open new stores.

Similarly to the visual exploration of time series, presented in the previous section, the visual exploration of top-k best spatial regions is currently implemented as a standalone application, and is planned to be integrated later into Visual Explorer.

We have proposed three scalable algorithms for the non-overlapping top-*k* BRS problem, which are detailed in Deliverable D3.2 (Section 2.2). The input parameters are the number of regions to be detected (*k*), the width and height of each region (*eps*), the attribute to be used in the scoring function (*targetColumn*), and two optional parameters for keyword filtering over the points (*keywordColumn*, and *keywords*). The output is the location of *k* non-overlapping regions with the highest score. The regions can be visualized on the map. Figure 26 shows the visualized output of a query over Italian companies where the user is detecting top-10 regions (*eps*=0.1 degrees) containing the highest number of any type of companies. As illustrated, the hotspots are mostly located at the center of Milan and Genoa. In Figure 27, the user has submitted the same query, but this time, he is asking for regions with the highest number of employees (*targetColumn* is set to employees attribute of the input data) where the company type is 'Industry,' 'Construction,' or 'Trade.' The visualized output implies that most of the employees working for specific companies are around Milan.

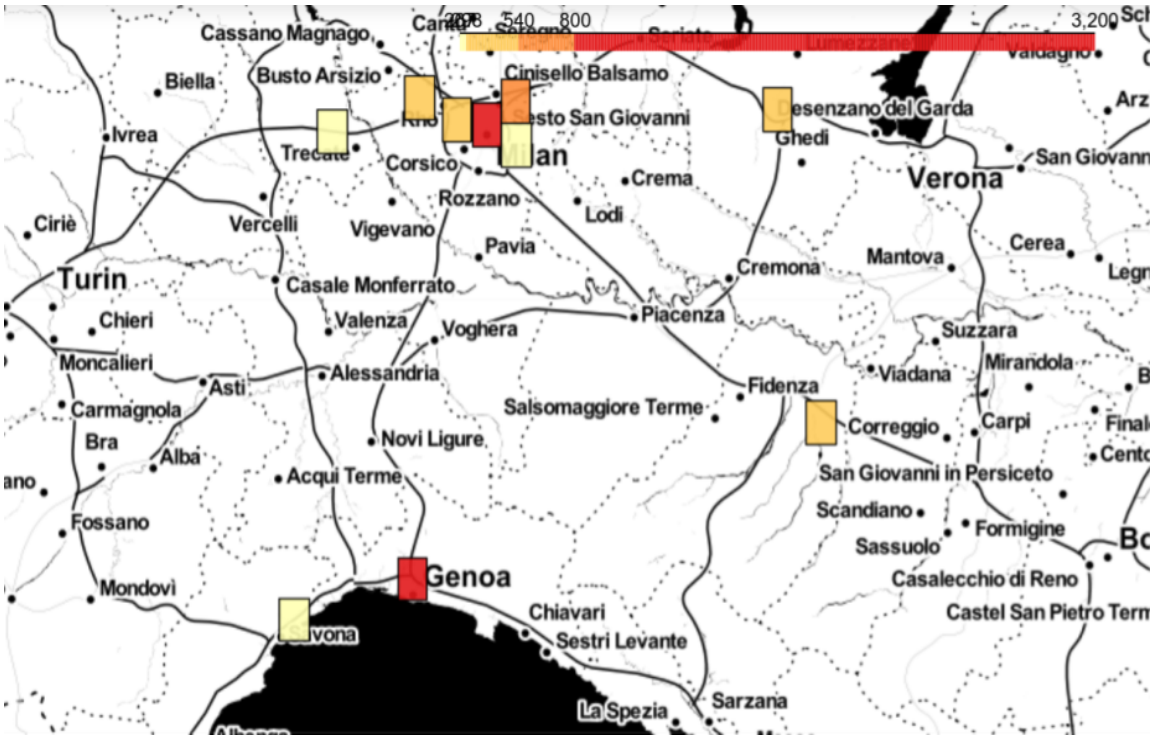

Figure 26 Top 10 non-overlapping regions (eps=0.1) containing the highest number of companies. The region color becomes darker as its score increases.

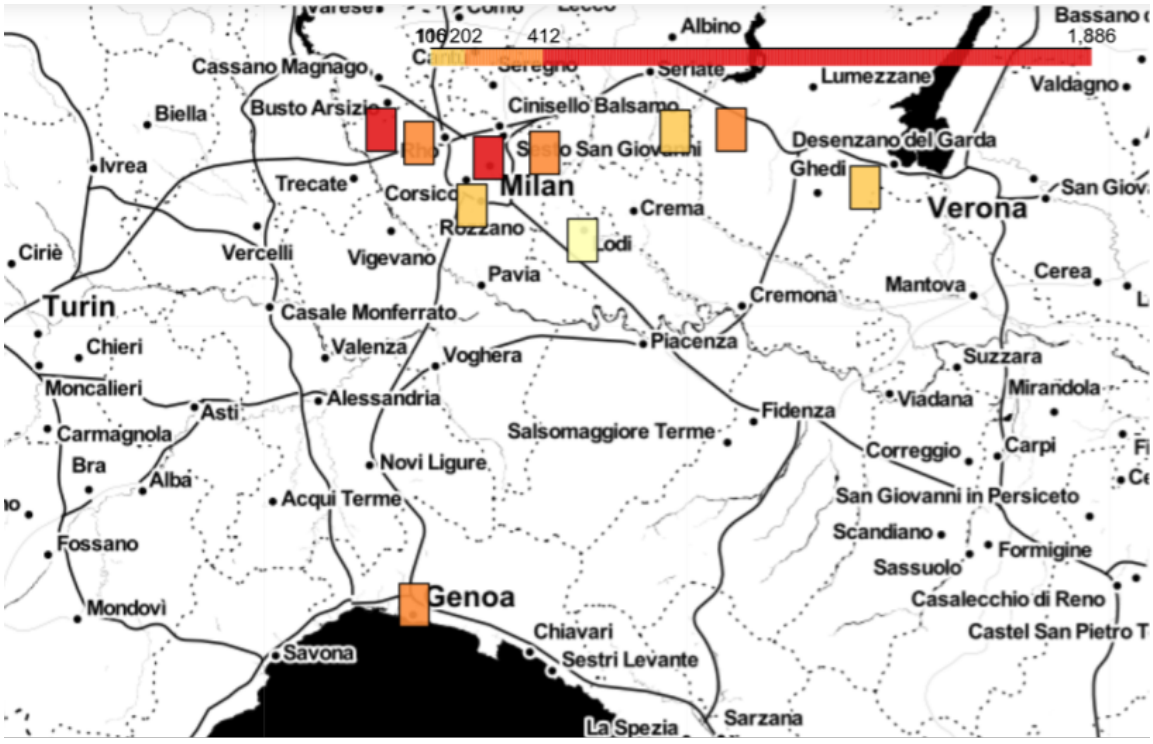

Figure 27 Top 10 non-overlapping regions (eps=0.1) containing the highest number of employees where the company keyword is 'Industry', 'Construction', or 'Trade'. The region color becomes darker as its score increases.

In Figure 28, the query is the same as in Figure 27, but the output regions are allowed to overlap each other. As shown, all the top 10 regions are located in the center of Milan with a slight distance to each other. This indicates that without support for non-overlapping results, the output regions are centered in just 1 or 2 areas, which is not sufficiently informative for the user. Thus, the ability of our algorithms to diversify the top-k computed regions is crucial in practice.

Using the above-described functionalities and visualizations, the user can specify different region sizes, keyword filters, and scoring attributes and visualize the results interactively to explore the best regions under different settings.

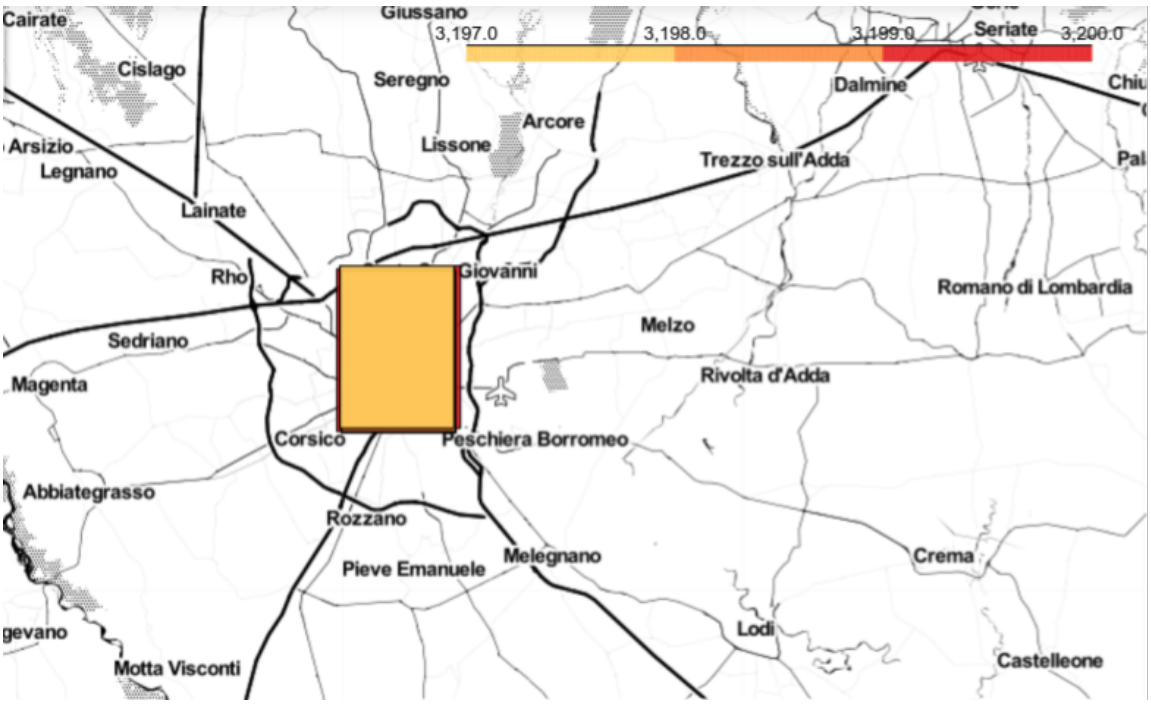

Figure 28 Top 10 overlaping regions (eps=0.1) containing the highest number of companies. Regions positioned over each other with small distance.

3.6 Panels for Other Upcoming Tasks

The design of SDL-Vis is tailored to support multiple, distinct analysis tasks on the same data foundation. Besides reducing the amount of displayed information and allowing to create task-specific interfaces, this also ensures maximal reusability and extendability of the visualization components. Therefore, throughout the project, additional visualization and interaction panels will be added for the upcoming tasks. Furthermore, existing panels will be extended with new functionalities, such as visual representations for geospatial data in the similarity search panel.

Link prediction, as well as community detection, both part of D3.3, will afford entirely new panels, supporting the abstracted visualization of the full heterogeneous information network or parts of

it. In both tasks, the user plays a crucial role in assessing the results of the automated algorithms and refine the model parameters according to his understanding of data and domain. Therefore, interactions with either the graph representation itself or abstract parameter refinement functionalities will have to be implemented in these panels.

Finally, change management, as part of D3.4, will afford abstract visualizations of the models of the SmartDataLake analysis pipeline as well as abstract measures over the heterogeneous information network. For continuous analytics, the analyst will be able to define custom metrics and track them over time, requiring a customized API to access both HIN and models of SmartDataLake.

4. Conclusions

In this report, we have presented our ongoing work on the visualization layer of SmartDataLake. This involves two main aspects: data provisioning by the Visual Analytics Engine, and data representation by the Visual Explorer. By defining a common API for the functionalities the Visual Analytics Engine offers, other visualization components can plug into different stages of the analysis process, enabling the iterative knowledge generation on the heterogeneous data sources of SmartDataLake. The Visual Explorer makes use of the exposed functionalities, providing task-driven visualization and interaction panels. It allows the refinement of the involved models and components, implementing the knowledge generation process in a unifying user-interface.

The interfaces and functionalities of the Visual Analytics Engine are described in detail. This includes data access and communication with the data virtualization component SDL-Virt and the data mining component SDL-HIN, as well as the internal data processing workflows of the Visual Analytics Engine. The Visual Explorer frontend application provides visualizations and interaction components to enable human knowledge generation on the provided data, models, and results.

While this document gives detailed definitions and schematics on the provided functionalities of SDL-Vis, the SmartDataLake analysis pipeline, together with the visualization layer, is still work in progress. Upcoming features and tasks will afford an extension of the existing data provisioning and data representation techniques. However, since the architecture of both the Visual Analytics Engine and the Visual Explorer is chosen to ensure maximum reusability and extendability, such changes can be directly embedded in the existing environment. Therefore, with upcoming tasks, the REST API of the Visual Analytics Engine will be extended with additional endpoints, which then will be accessed by newly created visualization and interaction panels of the Visual Explorer.

References

[1] F. Sperrle, J. Bernard, M. Sedlmair, D. Keim and M. El-Assady, "Speculative Execution for Guided Visual Analytics," *Machine Learning from User Interactions for Visualization and Analytics: An IEEE VIS 2018 workshop,* 7 8 2019.

[2] StackOverflow, *Developer Survey,* 2019.

[3] A. Satyanarayan, D. Moritz, K. Wongsuphasawat and J. Heer, "Vega-Lite: A Grammar of Interactive Graphics," *IEEE Transactions on Visualization & Computer Graphics (Proc. InfoVis),* 2017.

[4] M. Blumenschein, L. J. Debbeler, N. C. Lages, B. Renner, D. A. Keim and M. El-Assady, "v-plots: Designing Hybrid Charts for the Comparative Analysis of Data Distributions," *Computer Graphics Forum,* vol. 39, 2020.

[5] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing and J. , "Jupyter Notebooks - a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, 2016.

[6] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen and E. Keogh, "Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, 2016.

[7] W. S. Cleveland and S. J. Devlin, "Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting," *Journal of the American Statistical Association,* vol. 83, pp. 596-610, 9 1988.

[8] W. R. Tobler, "A Computer Movie Simulating Urban Growth in the Detroit Region," *Economic Geography,* vol. 46, p. 234, 6 1970.

[9] D. Keim, J. Kohlhammer, G. Ellis and F. Mansmann, Mastering The Information Age – Solving Problems with Visual Analytics, Eurographics Association, 2010.