Intelligent Querying for Implementing Building Aggregation Pipelines

1st Konstantinos Alexakis (K.A.)
DSS Lab of ECE School
National Technical University of Athens
Athens, Greece
kalexakis@epu.ntua.gr

2nd Panagiotis Kapsalis (P.K.)
DSS Lab of ECE School
National Technical University of Athens
Athens, Greece
pkapsalis@epu.ntua.gr

3rd Zoi Mylona (Z.M.)
Holistic S.A
Athens, Greece
zmylona@holisticsa.gr

4th Georgios Kormpakis (G.K)
DSS Lab of ECE School
National Technical University of Athens
Athens, Greece
gkorbakis@epu.ntua.gr

5th Evangelos Karakolis (E.K)
DSS Lab of ECE School
National Technical University of Athens
Athens, Greece
vkarakolis@epu.ntua.gr

6th Christos Ntanos (C.N)
DSS Lab of ECE School
National Technical University of Athens
Athens, Greece
cntanos@epu.ntua.gr

7th Dimitris Askounis (D.A)
DSS Lab of ECE School
National Technical University of Athens
Athens, Greece
askous@epu.ntua.gr

***Abstract*—Sensors, smart meters and IoT devices are key parts of the Building Information Systems. The amount of data generated from these sources is vast and the need for storage, fusion with secondary datasets (such as weather data) and aggregations has arisen in order to enhance building automation control activities. These data are stored on various data-sources, relational and non-relational, using different data formats. The combination of data coming from multiple data-sources constitutes a hard task, since each database uses different query language and structure. However, by combining all the available data-sources, it would be beneficial to reduce the volume of data during the training process. This paper presents an architecture that combines data from multiple data-sources (Databases, Object Storages, Building Information Systems) and create pipelines for aggregating the overall data.**

***Index Terms*—Data , Big Data, Query Engine, Data Processing, Energy Sector, Metadata**

I. Introduction

Nowadays, massive amount of data are generated during buildings' life-cycle. Building and energy information models have installed in their spaces: sensors, arduinos[1], Heating-Ventilation-Air Conditioning (HVAC) systems for receiving data related to their consumption, solar production, CO_2 emissions [1], [2]. Traditional systems receive this type of data and store them to relational databases without harmonizing, pre-processing and polishing them. This may result in the loss of meaningful information that potentially could be leveraged for improving building's energy consumption, CO_2 emissions reduction. The Key Performance Indicators (KPIs) calculation from data originated from Building Information Systems are of utter importance for controlling energy demands of buildings such as cooling or heating demand by proposing actions that could improve the building functionalities [3]. By combining Big Data State-of-the-Art solutions, it is possible to solve real-life problems surrounding buildings [4], [5]. Therefore, it is of crucial importance to understand the building data needs for creating smart systems that can handle and manage massive amounts of data loads and proceed to the next generation of building information solutions.

This work describes a Processing framework implemented in MATRYCS Ecosystem [2], which is connected through JSON [6] and AVRO[3] data schemas to Building information systems and receives data related to their consumption, CO_2 emissions, heating and cooling demand, building cadastrial information and other building metadata. All these different types are stored to a non-relational data storage for later processing.

This research contributes to the creation of a full enriched warehouse that collects data from Building Information Systems. Through JSON & AVRO connectors the raw building information is stored to MongoDB collections [7] and on top of them a query engine is responsible for connecting the raw building data and secondary information, which is pipelined in the system through the Staging Area. The output component of this work is the Data Feed Module, which utilises external connections (data temporarily stored in the Staging area) with raw building information, to apply transformations, such as duplicate removal, missing values management, outlier

[1]https://www.arduino.cc/

[2]https://matrycs.eu/

[3]https://avro.apache.org/docs/current/spec.html

detection. Furthermore, the calculation of pre-analytics will be conducted from Data Feed module in order to support the training of machine learning models over the enriched data. The intelligent querying mechanisms of the component will assure the creation of aggregation pipelines in order to reduce the computational load of calculations.

The rest of the paper is structured as follows: Section II analyses the architecture and implementation of the Data Feed module. Section III demonstrates a wide number of the functionalities of the Data Feed Module. Finally, Section IV presents the conclusions and the next steps.

II. Architecture and Implementation

In the Energy Sector, the adaption of sensors and IoT technologies has led to the unprecedented availability of Big Data [8]. The exploitation and storage of these data depends on the desired outcome of the service or application. For instance, a service may need near real-time monitoring of the energy consumption in a building, which means that the consumption data must be stored in a database that provides extremely high throughput data access. Another characteristic example would be a service that may need queries involving only few columns of the data with low latency, which can lead to the selection of a columnar database. The selection of one type of database is no longer possible to handle all the various services and applications. Therefore, taking the different needs of each service into account, the solution that is opted for in most cases is for the data to be stored in multiple databases.

The co-existence of multiple databases leads to complex and complicated data warehouses. In this way, the fetching of the data becomes a hard task, since the standard and the query structure differ among the databases. This paper proposes an architecture that deals exactly with this problem providing the capability of executing efficient queries to the databases as well as performing transformations and pre-processing techniques over the data.

The figure below depicts an overview of the proposed architecture, which consists of:

- Building Information Systems Connectors: Connectors used for receiving building data
- Staging Area: Object-Storage used for storing secondary data-sources, such as weather files, cadastrial data, etc.
- MongoDB Instance: Non-relational data storage used for hosting raw building information
- Query Engine: Presto[4] Instance used for Querying connected sources
- Data Feed Module: This module is leveraged for exposing data to MATRYCS upper layers. Its sub-modules, Query Builder and Data Pre-processing module, enable the fine-tuning, according to user input, of the granularity levels of processing and aggregations.

A. MATRYCS PROCESSING Data Storage Components

The MATRYCS Processing Data Storage System consists of a MongoDB Instance and more specifically for each different data schema a different collection was used. For this paper, we have utilised MongoDB, as main data storage, but initially, we had included CassandraDB to the Storage Layer. Both of them are non-relational databases, but the current solution supports only the MongoDB, because the implementation-integration with CassandraDB presented pitfalls and drawbacks due to the data format generated by the Building Information Systems.

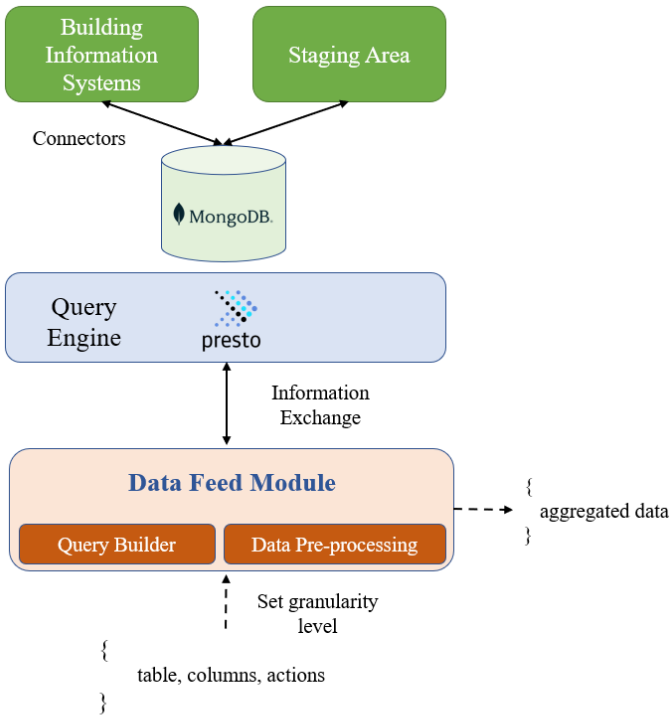

Fig. 1. High-level overview of the architecture

To begin with, MongoDB and CassandraDB [9] are popular databases that are used widely for the storage of data in the Energy Sector. These databases are characterised by the following features [10]:

- High-performance
- Scalability
- Schema-less NoSQL database
- Free and easy to be installed
- columnar / document oriented

During the first implementation phase of the Data Feed Module and Aggregation Pipelines in MATRYCS Project, the CassandraDB was utilised. However, due to the various formats of data presented in MATRYCS large scale pilots, the transition to a document database was an urgent need. This transition can be justified by the fact that the document-oriented databases support various data storage requirements without taking the schema of the data into consideration. The technical evaluation of storage requirements led us to the migration from CassandaDB to MongoDB.

[4]https://prestodb.io/

B. Migration from CassandraDB to MongoDB

During the MATRYCS-PROCESSING technology evaluation activities for the 1st technology release of MATRYCS Ecosystem, it was observed that columnar databases and in particular CassandraDB due to their nature cannot support nested data formats and updates on stored data, as CassandraDB is a schema-depended database. For that reason, databases that are schema-less and support the storage of nested object were investigated. The outcome of that procedure was that the MongoDB database is the ideal solution for keeping transformed and prepared data for Machine Learning/Deep Learning (ML/DL) training. Thus, MongoDB was selected instead of CassandraDB because it is flexible for document schemas, easily scalable and optimised for querying and analytics. Furthermore, MongoDB is the main data storage used in various FIWARE [5] components, that would possibly be integrated with MATRYCS Framework future releases.

Currently, all the components (Visualization Engine, Model Development Module, Serving & Evaluation Framework) that communicate with Data Feed Module receive data from MATRYCS MongoDB instance.

Data Migration was the main challenge to deal with, as it was necessary to transfer the existing data from CassandraDB to MongoDB, which was accomplished by using a series of Python scripts for receiving all the data from CassandraDB tables and then batch insert them to MongoDB. After data migration, the new version of the Data Feed has integrated Python functions for ensuring the connectivity and data exchange between MongoDB and the Data Feed REST (Representational state transfer) services.

C. Query Engine

The Building & Energy Management Systems expose time-series data from sensors that measure (e.g temperature, humidity, produced energy) and there is the need to manage this type of data [11]. Nowadays, enriched warehouses are multi-database systems that handle metadata databases and time-series databases. The emerging need is the management and querying of both data stores (real-time and metadata databases) and combine this stored information. A system that is database agnostic is needed in order to hide each database query language and let the end user to be capable to write SQL queries.

The current trend is the metadata of buildings' life-cycle as reported here [12]. Data querying solutions for building energy data are focused only on metadata querying and insights extraction. The proposed querying architecture in this research leverages a graph database that receives batch data in JSON format and then transforms them in graph entities. Furthermore the graph database persists ontologies and RDF schemas to enhance the stored metadata. By leveraging stored metadata patterns and relationships a REST API on top of the graph database receives JSON input and returns results from stored metadata. The main drawback of this data querying architecture is that it only takes building metadata into consideration.

In general, the goal of the cloud pipeline, in terms of storage, is to build an enriched data warehouse where raw building energy data are collected and queried via intelligent procedures, periodic tasks with an agnostic database manner. Furthermore, this data warehouse will provide the possibility to join external datasets with the stored information. For the implementation of the database agnostic data warehouse, the functionalities of the PRESTO query engine were leveraged. A data warehouse can be characterised as database agnostic, when it is able to query different databases, relational and non-relational, by using SQL queries. PRESTO is a distributed SQL query engine that provides interactive workloads by querying many different data-sources. In this case PRESTO is installed and configured on top of the MongoDB, enabling with this way the big data querying and analysis of the harmonised building data over a memory based architecture and without moving the aforementioned datasets to another structured system.

D. Data Cubes Implementation in Data Feed Module

Data Cubes are grouped data in a multidimensional matrix [13]. In Data warehouses, there is a large number of multi-dimensional data models, since the data are represented by multiple dimensions and multiple attributes. These multi-dimensional data are presented in the data cube as the cube represents a high-dimensional space [14]. The figure below demonstrates an example of Data Cubes.

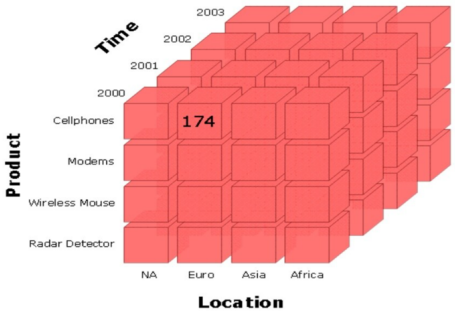

Fig. 2. Example of Data Cubes.

The Data Cubes are constructed thanks to the data combination from the Data Feed Module. In particular, the Data Feed Module joins data coming from the Staging Area and the MongoDB. This staging area has been implemented using a distributed file storage. The join between two different data-sources is achieved by leveraging the functionalities of the Query Engine, which utilises the different connection names of Presto. In that way, the join of two datasets coming from different datasources is feasible thanks to Query Engine.

The Figure 3 depicts the implementation of Data Cubes in the MATRYCS Data Feed module by leveraging the MATRYCS Query Engine, which is the mediator component

[5]https://www.fiware.org/

between the building information sources and the Data Feed Module. The Data Feed Module receives the data either from MongoDB collections or either MATRYCS Staging Area / Object Storage. Below the sub-component of Data Feed Module leverages the functionalities of Presto to join building data from MongoDB and Object storage. After the query calculation, the aggregation action is performed in order to expose aggregated data to MATRYCS upper layers. Finally, the Data Feed Module's REST API collection distributes the result data cube to MATRYCS Analytics applications. These data are also aggregated and prepared for Machine Learning training.

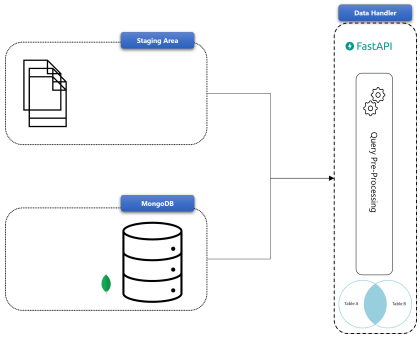

Fig. 3. Data Cubes implementation in Data Feed Module

E. Data Feed Module Architecture

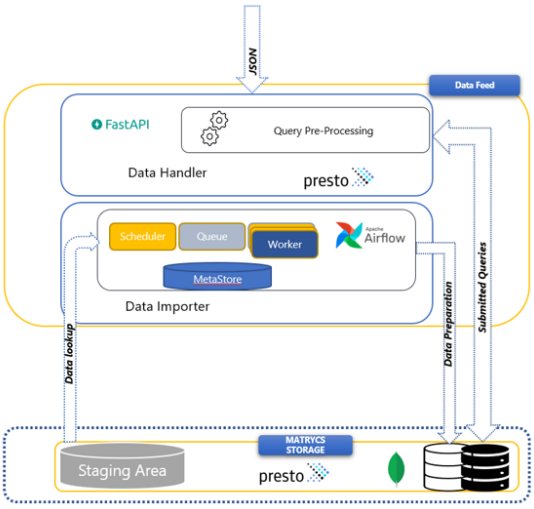

Fig. 4. Architecture of the Data Feed Module.

The Data Feed Module is the component responsible for data transferring between MongoDb Instance and Machine Learning Training Procedures. More specifically, it consists of two sub-components: The Data Importer and the Data Handler. The architecture of the Data Feed Module is presented in the Figure 4.

The Data Importer is a data pipeline system that receives data from MATRYCS distributed storage. When the data are on boarded to the Data Importer, a series of basic data preparation steps are conducted over them, such as the preparation of the data to be inserted on the MATRYCS MongoDB and the deletion of duplicate and null values. The Apache Airflow has been selected for being the basis of data pipelines management on Data Feed Module. It contains the following core components:

- Web Server: This is the UI of Airflow that can be used to get an overview of the overall health of different Directed Acyclic Graphs (DAG) and also in visualizing different states of each DAG .
- Scheduler: This is the most important part of Apache Airflow, as it orchestrates various DAGs taking care of their interdependencies.
- Executor: Executors are the components that actually execute tasks. The type of Executor used in production is the CeleryExecutor, based on Celery framework, for scaling the execution of tasks/DAGs across computational resources.
- Meta-data database: This database stores metadata about DAGs and Apache Airflow configuration details.

The data are inserted to MATRYCS-PROCESSING through Data Importer subcomponent and stored to MongoDB.

The Data Handler is the REST service responsible for distributing these stored data across MATRYCS-PROCESSING components. This module leverages Python [15] libraries such as Presto Python Client, Pandas Framework [16] for DataFrames and FastAPI [6] for REST services. One core functionality of the Data Handler is the building of the SQL queries (Query Builder). These REST APIs are used for enabling data selection, data aggregation, data grouping, dates handling, numerical scaling, categorical encoding (one-hot encoding, label encoding) and converting time series to supervised procedures. The Data Handler is connected with MATRYCS Query Engine (Presto), and all queries are transformed into SQL queries and all the MongoDB collections are transformed into tables in Presto.

The Data Importer waits for incoming data on MATRYCS Storage Staging Area. The Staging Area is a distributed file storage where data are placed after the execution procedures of Building Information System. The preparation pipelines are scripts developed in Python 3.7 which have been integrated on Airflow [7] framework, and apply transformation steps. For example dropping duplicates, removing null values, and normalising dates. These actions are scheduled and executed from Apache Airflow workers, and they are executed when new data are detected. At the end, the transformed data are inserted to MATRYCS Storage for later use.

The Data Handler is the collection of REST services responsible for distributing the data across MATRYCS-PROCESSING. These services receive JSON payloads which are processed from Query pre-processing class of the Data Handler for constructing the Presto-SQL query. These queries

[6]https://fastapi.tiangolo.com/
[7]https://airflow.apache.org/

are sent to MATRYCS Storage for getting the results back as a response. Furthermore, these APIs are used for data selection, data aggregation, data grouping, timeseries transformation and could be the input for multiple MATRYCS components such as Visualization Engine, Serving framework, etc.

III. Demonstration

The Data Feed Module is the component responsible for transferring the stored prepared information from MongoDB and creates cubes of data in order to be used from MATRYCS Analytics layers and applications. The implementation of these structures combines information derived from Building information systems and performs aggregations that enhance the quality of predictions that will improve the building life-cycle .

Aggregations that are applied are min-max scaling, categorical encoding, table joins, grouping of tables and selections over stored data. The following tables contain some information that demonstrate Data Feed Module capabilities. Query Engine is the component responsible for the combination and convergence of stored MongoDB collections and raw building data stored in MATRYCS object and file storages. Presto provides capabilities, through its catalogues of registered storages, can apply aggregations over different data sources that contain building functional information.

To be more specific, the following tables contain various API calls, that are sent to the Data Feed Module, which constructs SQL queries, using the Query Builder. As next step, the Data Feed Module communicates with the Query Engine, which retrieves the data from the various data sources and make them available. In that point, the constructed SQL query is executed and the Data Feed Module returns the desired outcome. For the needs of the demonstration of our work the datasets coming from a large scale MATRYCS Pilot will be used. These datasets contain energy data (energy consumption, demand etc.) as well as information about events.

In the table below, the following query is used for receiving records with ”timestamp” and ”value” (energy consumption) fields from 01-12-2020 to 02-12-2020.

TABLE I
Demonstration of Data Feed Module (SELECT)

Description	Functionality

Select data from a dataset	POST /complex/select/query HTTP/1.1
	
	Host: hostname:8000
	Content-Type: application/json
	{
	”table”: ”pilot”,
	”columns”: [”timestamp”, ”value”],
	”where_column”: ”timestamp”,
	”between_values”: [”2020-12-01”, ”2020-12-02”]
	}

The Table II presents a query that includes aggregation functions (MAX, AVG) about the ”value: (energy consumption) of the MATRYCS pilot.

TABLE II
Demonstration of Data Feed Module (AGGREGATIONS)

Description	Functionality

Apply aggregation functions to a dataset	POST /complex/select/query HTTP/1.1
	
	Host: hostname:8000
	Content-Type: application/json
	{
	”table”: ”pilot”,
	”aggregation_metrics_list”: [”MAX”,”AVG”],
	”aggregation_columns_list”: [”value”, ”value”],
	”aggregation_metrics_alias_list”: [”max_value”, ”avg_value”]
	}

The Table III presents a group by query, which returns the average value of the ”value” grouped by year in descending order.

TABLE III
Demonstration of Data Feed Module (GROUP BY)

Description	Functionality

Group data by column	POST /complex/group/query HTTP/1.1
	Host: hostname:8000
	Content-Type: application/json
	{
	”table”: ”pilot”,
	”grouping_columns”: [”year”],
	”aggregation_metric_alias”: ”avg_year_value”,
	”aggregation_metric”: ”AVG”,
	”aggregation_column”: ”value”,
	”order_by_column”: ”year”,
	”order”: ”DESC”
	}

The table IV presents a query that joins two tables, ”pilot” and ”pilot_calendar”, and filters the data in order to return the records that refer only to the first eight months of the year.

TABLE IV
Demonstration of Data Feed Module (JOIN)

Description	Functionality

Join two tables	POST /complex/join/query HTTP/1.1
	Host: hostname:8000
	Content-Type: application/json
	{
	”join_tables”: [”pilot”, ”pilot_calendar”]
	”columns_table1”: [”value”, ”timestamp”, ”month”],
	”columns_table2”: [”event”],
	”join_types”: [”type”: ”INNER”, ”table1”: ”pilot”, ”column1”: ”month”, ”table2”: ”pilot_calendar”,”column2”: ”month”],
	”where_symbol”: ”¡”,
	”where_column”: ”month”,
	”where_table”: ”pilot”,
	”where_clause_term”: ”9”
	}

IV. Discussion and Future work

This paper presented the Implementation of an architecture for building aggregation pipelines. Using the capabilities and functionalities of Presto, the system is able to process, apply aggregations and pipeline transformations on building data and metadata. In this way, all the components are able to receive enriched data that can lead to insightful outcomes and predictions, which may mitigate or even solve real-life problems of the buildings (energy consumption, CO_2 emissions etc.).

As future work, the presented work will be integrated in the MATRYCS Ecosystem and it will be extended in order to receive as additional input streaming data. These data may be energy consumption, temperature, etc, that are measured by sensors. This extension will facilitate the solution of near-real time problems.

Acknowledgment

This work has been funded from the European Union's Horizon 2020 research and innovation programs under the MATRYCS project 'Modular Big Data Applications for Holistic Energy Services in Buildings', grant agreement No 101000158, and the 'Multidisciplinary Expert System for the Assessment & Management of Complex Brain Disorders' (MES-CoBraD) project, grant agreement No 965422.

References

[1] V. Marinakis, "Big data for energy management and energy-efficient buildings," *Energies*, vol. 13, no. 7, p. 1555, 2020.

[2] V. Marinakis, H. Doukas, J. Tsapelas, S. Mouzakitis, Á. Sicilia, L. Madrazo, and S. Sgouridis, "From big data to smart energy services: An application for intelligent energy management," *Future Generation Computer Systems*, vol. 110, pp. 572–586, 2020.

[3] J. Al Dakheel, C. Del Pero, N. Aste, and F. Leonforte, "Smart buildings features and key performance indicators: A review," *Sustainable Cities and Society*, vol. 61, p. 102328, 2020.

[4] V. Marinakis, C. Karakosta, H. Doukas, S. Androulaki, and J. Psarras, "A building automation and control tool for remote and real time monitoring of energy consumption," *Sustainable Cities and Society*, vol. 6, pp. 11–15, 2013.

[5] Y. Wei, X. Zhang, Y. Shi, L. Xia, S. Pan, J. Wu, M. Han, and X. Zhao, "A review of data-driven approaches for prediction and classification of building energy consumption," *Renewable and Sustainable Energy Reviews*, vol. 82, pp. 1027–1047, 2018.

[6] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, "Foundations of json schema," in *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2016, pp. 263–273.

[7] S. Bradshaw, E. Brazil, and K. Chodorow, *MongoDB: the definitive guide: powerful and scalable data storage*. O'Reilly Media, 2019.

[8] V. Marinakis and H. Doukas, "An advanced iot-based system for intelligent energy management in buildings," *Sensors*, vol. 18, no. 2, p. 610, 2018.

[9] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[10] V. Abramova and J. Bernardino, "Nosql databases: Mongodb vs cassandra," in *Proceedings of the international C* conference on computer science and software engineering*, 2013, pp. 14–22.

[11] V. Marinakis, C. Karakosta, H. Doukas, S. Androulaki, and J. Psarras, "A building automation and control tool for remote and real time monitoring of energy consumption," *Sustainable Cities and Society*, vol. 6, pp. 11–15, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210670712000467

[12] P. Kapsalis, G. Kormpakis, K. Alexakis, and D. Askounis, "Leveraging graph analytics for energy efficiency certificates," *Energies*, vol. 15, p. 1500, 01 2022. [Online]. Available: https://www.mdpi.com/1996-1073/15/4/1500

[13] V. Harinarayan, A. Rajaraman, and J. D. Ullman, "Implementing data cubes efficiently," *Acm Sigmod Record*, vol. 25, no. 2, pp. 205–216, 1996.

[14] S. Sarawagi, R. Agrawal, and N. Megiddo, "Discovery-driven exploration of olap data cubes," in *International Conference on Extending Database Technology*. Springer, 1998, pp. 168–182.

[15] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[16] W. McKinney *et al.*, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, vol. 445. Austin, TX, 2010, pp. 51–56.

Author Contributions

Conceptualization, Z.M., K.A., P.K.; methodology, Z.M., K.A. and P.K.; software, P.K., Z.M., G.K., E.K. and K.A.; validation, P.K., Z.M., G.K., E.K. and K.A.; formal analysis, P.K., Z.M., G.K., E.K., K.A. and C.N.; investigation, P.K., Z.M., G.K., E.K., K.A. and C.N.; writing—original draft preparation, K.A. and P.K.; writing—review and editing, D.A., K.A., P.K. and Z.M.; supervision, D.A., C.N. and Z.M.; funding acquisition, D.A., Z.M. and C.N.. All authors have read and agreed to the published version of the manuscript.