Linked Data for Smart Neigbhorhood

2023

LINKED DATA FOR SMART NEIGHBORHOOD

MAKING URBAN ENERGY USE MORE MEANINGFUL USING SEMANTIC DIGITAL TWINS

2023/053

University of Technology Eindhoven
Smart Buildings and Cities
Royal KPN N.V.

S.R. (Sander) de Meij s.r.d.meij@tue.nl

Dr. D. (Dujuan) Yang Ir. A.J.A. (Alex) Donkers
Drs. Ing. M. (Matthijs) Klepper Dr. F. (Frank) Mertz

Thesis Evaluation Committee

Scientific Supervisor	Dr. D. (Dujuan) Yang
Scientific Supervisor	Ir. A.J.A. (Alex) Donkers
Company Representative	Dr. F. (Frank) Mertz
First Independent Member	Rik Budel
Second Independent Member	Dr. G.Z. (Gamze) Dane
Other Member	Prof. Dr. Ir. B. (Bauke) de Vries
Chair of the Committee	Dr. Ir. A.D.A.M. (Astrid) Kemperman

The design described in this thesis has been carried out in accordance with the TU/e Code of Scientific Conduct

Foreword

This thesis marks the end of my eight-year period at the University of Technology Eindhoven. What started as an ambition to become an architect has ended in me entering the niche of Linked Data in the Built Environment. The last two years have been an amazing opportunity to pivot toward a topic I find truly interesting and captivating. Developing the skills to handle, process, and make data more meaningful has been a wonderful experience. I genuinely believe I've learned more about being an engineer in the last two years, than in the entire 6 years before that. This is not least due to the incredible supervision I've received from Dujuan and Alex, who I would like to thank for their insights, feedback, and lengthy discussion about varying topics. Your honesty and expertise have always motivated me to put in a little bit more effort. I would like to thank you for the part you've had in me becoming the engineer and person I am today. Moreover, discussions with Matthijs were always insightful and enjoyable, being able to see the problem from a different perspective, while showing enormous enthusiasm, forced me to think in different ways. Which, in my opinion, has elevated the project to another level. Lastly, I would like to thank Frank for getting to know the project so quickly and being able to provide guidance during the closing phases of the project. Thank you all. Furthermore, I would like to thank all my colleagues on the 9th floor, who I hope will remember to go for lunch at 12 o'clock sharp. While some people might consider lunchtime a time for relaxation from work, our lunch sessions were deeply philosophical discussions about life, culture, and existence in general (most of the time anyway). However, they were also some of the most enjoyable moments during the last two years. Finally, I think I owe some thanks to my family, who were able to suffer through my eternal ramblings about graphs, data, and dashboards. I think they might be qualified as engineers by now as they must have sat through my speeches for hundreds of hours. Of course, no one deserves more appreciation than Amy who has motivated me through the last two years, listened to all my ramblings and complaints but was always able to be interested. Most importantly, she taught me the most important lesson of the last two years which is to not only do what you love but to do with a smile.

Sander de Meij

Summary

Cities increasingly contribute to overall energy consumption, which leads to a large set of challenges. However, there is only limited integration of energyrelated data on an urban level. This limits possibilities for cross-domain monitoring, simulation, and intervention. This project aims to adapt semantic web technologies to integrate cross-domain data and information on multiple scales, while also striving to develop a visualization to allow stakeholders to interpret and work with the data more intuitively. This would be a first step towards a semantic digital twin of the city under investigation (Eindhoven, the Netherlands). This semantic digital twin is based on Semantic Web Technologies and Linked Data, which is a method to transform data into structured graphs that allows for the integration of data on multiple levels. While this project is not the first of its kind to address urban challenges using Semantic Web Technologies, it tries to improve on existing ontologies that lack critical features that are deemed necessary for this project. Therefore, a new ontology is created, named the Neighborhood Energy Ontology (NEO). NEO described urban areas as neighborhoods that can have certain properties. These properties are related to the way they are measured and the time frame over which they are relevant. Moreover, neighborhoods can contain other neighborhoods, and can therefore inherit properties from other neighborhoods. To summarize, neighborhoods of varying scales may possess multiple properties that can be measured or assessed through multiple executions with different procedures. These executions may yield multiple results, each with a designated time interval that reflects its temporal relevance. This ontological structure is used to create the previously mentioned semantic digital twin; Neo Dash. The goal of this dashboard is to allow the user to meaningfully explore the available data, and assess relevant challenges. The dashboard consists of four main functions: the map, query, table, and graph. The map visualizes the results of queries and analyses where possible. Initially, the map shows all neighborhoods of the city of Eindhoven, which can be explored manually by the user. The main interactive part of the dashboard is the query functionality which allows the user to build (compound) queries based on the properties that are available in the database. The dashboard uses the defined metadata to provide the user with the most suitable query options as numerical properties require different query functionalities than categorical or nominal values. While the map represents the queried results visually, the table represents the results more classically. In contrast, the graph shows the relationships of the found results with other aspects of the data. For instance, if the queried property shares measurement methods with other available properties. Combined, these functionalities aim to provide more meaningful data exploration and discovery than currently available methods. Lastly, to show the added value

of this semantic data structure and the created digital twin several use cases are explored. Firstly, several definitions of energy poverty are explored. As no one definition of energy poverty is available, the three most common definitions are represented using a combination of the map and a new graph representation visualizing the household income relative to the estimated cost of energy for that neighborhood. Moreover, the most recent definition includes the energetic quality of the household dwelling (energy label). Therefore, the analysis includes this building-level data as well. This use case shows how data from disparate sources can be combined using this method, and meaningfully represented using the created digital twin. The user is able to explore this issue from multiple perspectives and interact with the data in a novel way. Secondly, it is assessed how more temporally accurate data can be incorporated into the ontological structure previously discussed. Randomly generated energy use data is generated for the University of Technology campus' buildings, which shows how this type of data can also be explored using the proposed digital twin. Moreover, it is assessed how such data could practically be collected and monitored using existing technologies. This use case shows how individual or property managers can also benefit from this type of development. Thirdly, while the available data is rather extensive, some missing parts do create challenges in the first use case. Therefore, it is explored how Linked Data can be used as a basis for predictive analytics by estimating energy label categories for buildings that have no energy label available. Using a statistical model, it is estimated whether a building's energy label is above or below C (labels range from A++++ to G). This model is able to perform this task with 89% accuracy (92% precision and 92% recall), which is deemed sufficiently accurate for the purposes of this project. This model is used to supplement the missing data when the user investigates individual data. Moreover, the energy poverty analysis which has been described previously is enriched using these categories, resulting in a more complete estimation of the overall scope of the issue for the city under investigation. The goal of this project was to provide more meaningful urban data using Semantic Web Technologies and Linked Data and create a digital twin on top of this data structure. The created semantic digital twin and underlying ontological data structure is a good first step towards this goal. The use cases show that more meaningful data can indeed be provided and that more a more holistic and insightful analysis can be performed using the proposed solution.

Content

Acronyms

API	Application Programming Interface
BIM	Building Information Model
BOP	Building Performance Ontology
BOT	Building Topology Ontology
CWA	Closed World Assumption
EM-KPI	Energy Management Key Performance Indicator Ontology
GUID	Globally Unique Identifier
IoT	Internet of Things
KPI	Key Performance Indicator
LiDAR	Light Detection and Ranging
LIHE	Laag Inkomen, Hoge Energierekening
LILEK	Laag Inkomen, Lage Energetische Kwaliteit
MQTT	MQ Telemetry Transport
NEO	Neighborhood Energy Ontology
OWA	Open World Assumption
RDF	Resource Description Network
RDFS	RDF Schema
RF	Random Forest
SAREF	Smart Applications REFerence Ontology
SPARQL	SPARQL Protocol and RDF Query Language
URI	Unique Resource Identifier

Figures

1. INTRODUCTION

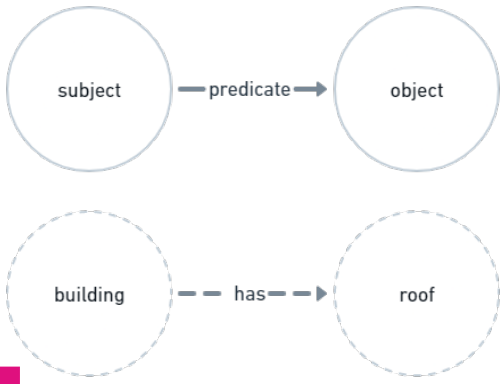
Growing urban populations (United Nations Department of Economics and Social Affairs, 2019) cause increased energy consumption in cities. As a result, cities currently consume 'two-thirds of primary energy resources and are responsible for more than 70% of Green House Gas emissions worldwide' (Abbasabadi et al., 2019). Buildings in these cities account for 40% of global energy consumption (Corry et al., 2015). To address this issue, there is great potential in urban energy modeling which can result in increased energy use efficiency on an urban and building level (Ali et al., 2021). However, according to Curry et al. (2013), there is limited integration of traditional building information and other data, such as energy consumption. This limits possibilities for cross-domain monitoring, simulation, and interventions. More readily available information could indeed facilitate the identification of problems and solutions concerning urban energy consumption (Ali et al., 2020). Moreover, while many studies focus on integrating data on the building scale (Ali et al., 2019; Corry et al., 2015; Curry et al., 2013; Degha et al., 2019), most goals for reducing energy use and Green House Gasses are set on a national level, and most action is taken at the city scale (Li et al., 2017). This project aims to adapt semantic web technologies to integrate crossdomain data and information on multiple scales, while also striving to visualize this integration to allow stakeholders to interpret and work with the data more intuitively. Placing this effort in the current digital twin paradigm, the definition provided by VanDerHorn and Mahadevan (2021) should provide some insight, as they describe a digital twin as 'a virtual representation of a physical system (and its associated environment and processes) that is updated through the exchange of information between the physical and virtual systems.' This project aims to take the initial steps toward such a digital twin. To achieve this goal semantic web technologies (section 1.1) will be implemented to take the first steps toward an interactive, semantic Digital Twin of the city of Eindhoven (the Netherlands).

1.1 State of the Art

Regarding traditional urban energy use modeling, Abbasabadi and Mehdi Ashayeri (2019) provide an overview of the currently available categories of modeling. First of all, the authors separate urban energy use models between top-down and bottom-up, where 'top-down models examine cities at a macro scale. They are not concerned with individual end-uses; rather, they treat the built environment as an energy user and utilize historical aggregated energy data to understand how energy is used in cities'. Conversely, bottom-up models 'localize energy use studies and considers urban attributes at the microscale of individual units, i.e. individual buildings or a collective set of buildings'. Furthermore, the distinction

is made between data-driven (statistical) and simulation models, within the bottom-up approach. The authors conclude that to date, only a limited number of tools exists to estimate energy use in urban or neighborhood contexts in an integrated manner and that existing tools such as CitySim, EnergyPlan, E-GIS, Urban Building Energy Models (UBEMs), Urban Modeling Interface (UMI), and City Building Energy Saver (cityBES), rely on the estimation of the energy use at city scale through a GIS-based platform, 2D GIS and/or 3D GIS using CityGML (Abbasabadi and Mehdi Ashayeri, 2019).
Current urban modeling tools, therefore, are generally not capable of assessing energy use in an integrated manner, where data from multiple scales (top-down and bottom-up) and domains can be integrated. As an alternative to more traditional urban (energy use) modeling, Semantic Web Technologies and Linked Data are explored as a possible solution. However, the concepts of 'Semantic Web' and 'Linked Data' need some explanation as well as some definitions. Moreover, already existing research within this field needs to be addressed. In short, 'semantic web technologies [...] allow to represent information in structured graphs and efficiently integrate (building) information of an entirely different nature. As a result, the development of software applications that rely on multiple information sources is in reach' (Pauwels et al., 2017). The important element in this definition is the 'structured graphs', at the core of these graphs 'stands a flexible and generic language that allows to easily represent and combine information from diverse knowledge domains, namely RDF. The semantic web thus becomes a semantic network in which information is represented as directed labeled graphs (RDF graphs)' (Pauwels et al., 2017). Here, it becomes clear that the structured graphs at the core of the semantic web are defined by the Resource Description Network (RDF), moreover, these RDF graphs are both labeled and directed. The idea of such a graph is represented in Figure 1, where 'each node in such a graph represents a concept or object in the world, identified with a Unique Resource Identifier (URI)' (Pauwels et al., 2017). As can be seen in the example of Figure 1, each trio (triple) is given by a 'subject', 'object', and 'predicate', where each element is represented by an URI describing something in the world. This example states that the subject 'building' is connected to the object 'roof' by the predicate 'has', i.e. this building has a roof. What exact building this is or what specific roof it has can be defined by the URI linking to a unique resource defining the instance. Moreover, in the next linkage, the 'roof' can become the subject and be linked to a third object, and so forth. 'By describing all information as such interlinked directed labeled graphs, a uniform representation of information is achieved, making information reusable by both humans and computer applications' (Pauwels et al., 2017). Moreover, translating the RDF principle to its most basic form: 'The most basic elements describing such ontologies are contained in the RDF Schema (RDFS) vocabulary, which consists of the specifications of classes, subclasses, comments, and data types. An RDFS interpreter is able to infer extra RDF statements that are implicitly available via the RDFS constructs' (Pauwels et al., 2017). In the example below, a 'building element' class could be defined and attributed to the 'roof' object, indicating that a 'roof' is (at least) a building element. These structures of classes and relationships are combined into ontologies, which can be considered predetermined semantic structures. Within the semantic web, an important aspect is the Open World Assumption (OWA). This assumption states that if the model does not specify something,

it is neither necessarily false nor true. The example below does not specify that the building has a door. In a Closed World Assumption (CWA), this would mean that this building does not have a door (this is the case in traditional data and information technologies regarding buildings, e.g. Building Information Model (BIM)). In the OWA, however, this is not the case and the building might, or might not, have a door. The term 'semantic web' was coined by Tim Berners-Lee in 2001 and was quite visionary as it included all features in the semantic web stack. The term 'linked data', on the other hand, was coined in 2006, also by Tim Berners-Lee, in response to the finding that quite some data was being published on the web, seemingly following the semantic web idea but actually never linking to outside data, and thus in fact not realizing the initial core idea behind the semantic web, which is linking data. Therefore, Berners-Lee laid out four rules that need to be followed to obtain linked data truly. These have by now evolved into the five stars of linked data (Pauwels et al., 2017). These five stars are defined as follows (Hausenblas and Kim, 2012): 1) Make your stuff available on the web (whatever format) under an open license. 2) Make it available as structured data (e.g. Excel instead of an image scan of a table). 3) Make it available in a non-proprietary open format (e.g. CSV instead of Excel). 4) Use URIs to denote things so that people can point at your stuff. 5) Link your data to other data to provide context. This five-star system can be used as a measure of how well your data adheres to the principles of linked data as proposed by Tim Berners-Lee, where it is suggested to strive for as many stars as possible. This research will aim to achieve five stars, and link data to its appropriate context. This project is not the first to address the challenges described earlier, therefore, it



F.1 Basis of RDF graphs

is necessary to review a sample of already existing research. Regarding already existing ontological structures relevant to this project, the study by De Nicola and Villani (2021) gives a preliminary overview of available ontologies related to several urban topics. Regarding energy use, they identify several ontologies, however, they are considered unsuitable for this research. They can not describe urban data on differing urban levels, as they mostly relate to other urban units like microgrids (Chun et al., 2020), houses (Reinisch et al., 2011) or appliances (Daniele, 2020). Moreover, the authors give an overview of available ontologies describing urban systems, which are not aligned with the specific goal of this project as they mostly describe specific urban infrastructure or three-dimensional geospatial objects. Besides these ontologies, SAREF4CITY (Poveda-Villalon et al., 2020) can be considered. This ontology focuses on extending Smart Applications REFerence Ontology (SAREF) (Daniele et al., 2015) to create a common core of general concepts for smart cities and data-oriented to the Internet of Things (IoT) field (Poveda-Villalon et al., 2020) and describes a data structure that allows for the description of several city objects, their geographical definition and corresponding measurements. While this ontology does

reflect the core idea of this project, it is deemed unsuitable, as it is more aimed at structuring Internet of Things (IoT) data and Key Performance Indicator (KPI) measurements. Moreover, this project tries to capture urban data, without relying on a geographical definition as they are considered hard to work with and impractical for the implementation suggested in this project. Secondly, the Energy Management Key Performance Indicator Ontology (EM-KPI) can be considered. This ontology is created to describe the relationship between the master data sources for identifying energy performance problems and key areas for improvement and to help energy managers make informed decisions regarding energy efficiency measures (Li et al., 2019). Again, this ontology has similar goals as this project, however, is deemed unsuitable for the purposes of this project as it has an extensive focus on KPI measurement, energy systems, and building aspects, while this project is focused on purely urban data. Moreover, as will be explained below, the central concepts of the ontological structure created in this project is based on the Building Performance Ontology (BOP) (Donkers et al., 2021) and Building Topology Ontology (BOT) (Rasmussen et al., 2020). These ontological structures are considered to be able to describe building-level information and data in high detail and are therefore extended in this project. As the previously described ontologies are not connected to either BOT or BOP, they are not reused in this project. As mentioned, this project is not the first to address energy-related challenges on an urban (or building) scale. As will become clear in section 4, energy poverty (section 4.1) is a long-standing issue with several analyses done in multiple countries (Mulder et al., 2023; Department for Business, 2020; Department for Energy Security & Net Zero, 2023). Similarly, the prediction of energy consumption (and energy labels, section 4.3) is a rich field with large amounts of research being done (Zhao and Magoul`es, 2012; Kim and Cho, 2019; Wang et al., 2021; Kolter and Ferreira, 2011; Amber et al., 2015). However, what these studies do not investigate is the potential of implementing Semantic Web Technologies in this context, in connection to a digital twin. Making this type of data more interactive, easily accessible, and meaningful to the user of this data.

1.2 Problem Statement

It has been described how cities and the built environment contribute significantly to the energy demand of the world, which causes a multitude of challenges. Where traditional urban energy use models can be categorized as topdown or bottom-up, these challenges need a more holistic approach where a range of spatial and temporal scales can be analyzed structurally. A lack of connected and meaningful data is seemingly a barrier to this approach. Therefore, Semantic Web Technologies seem an appropriate solution, however, no suitable development has been found. This project will, therefore, aim to create a semantic structure that is capable of describing urban (energy) data on multiple spatial and temporal scales (section 2), in order to gain insight into these challenges. Moreover, this structure will be designed to be highly practical in use (section 3) and applicable in multiple settings (section 4). The practicality of the data structure will be explored through an interactive dashboard (semantic digital twin) which will show the implementation of several use cases.

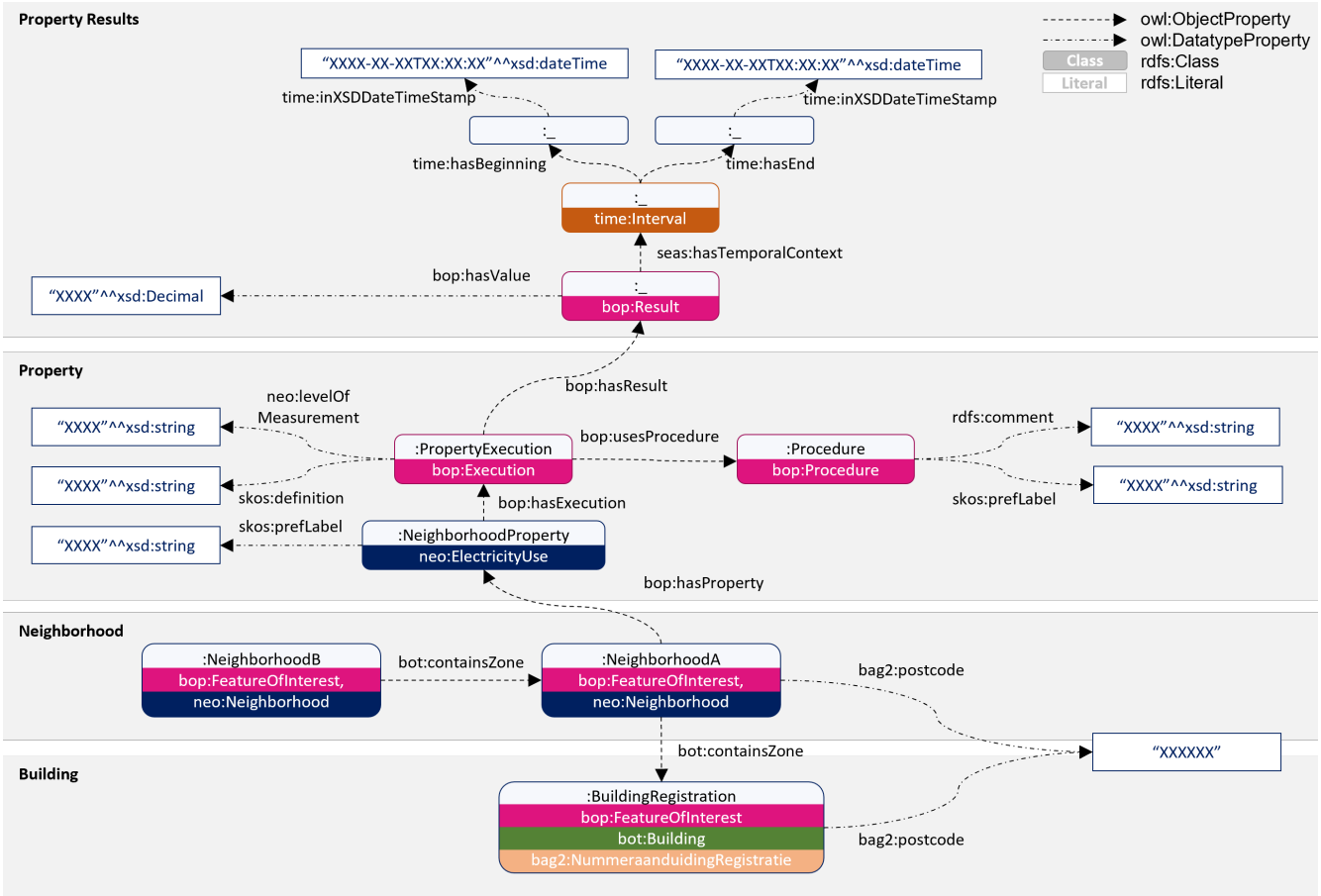
2. INTEGRATION

Taking into account the ontologies described in section 1.1, a new ontological structure has been created named Neighborhood Energy Ontology (NEO), which reuses and extends multiple existing data structures. NEO tries to achieve the previously described goals by defining 'neighborhoods' as urban areas, which can contain other urban areas of a different (smaller) scale. These neighborhoods are linked to certain properties that are attributable to these areas, following a similar structure as defined in the Building Performance Ontology. In this project, neighborhoods are considered a 'bop:FeatureOfInterest' and therefore can be associated with a 'bop:Property'. This structure is given in Figure 2 (namespaces are defined in Table 1). In this overview, it is shown how neighborhoods can contain other neighborhoods, of a different scale. NEO can therefore describe data on multiple levels, where a contained neighborhood might be assumed to inherit the properties described by the containing neighborhood. Moreover, multiple properties can be attributed to a neighborhood, which might come from different domains. Therefore, a more holistic description of urban data can be given. To create a high-level structure of these properties, the property structure in Figure 3 is adopted. This figure shows that a 'neo:Neighborhood' is a sub-class of a 'bot:Zone', which allows for the previously described relationship where neighborhoods (zones) can contain other neighborhoods (zones). Moreover, neighborhoods can thus contain buildings (bot:Building) as shown in Figure 2. An anchor point for building information in NEO is the existing data structure of the Cadastre, Land Registry and Mapping Agency of the Netherlands (Dutch: Kadaster). Their Key Register for Addresses and Buildings (Dutch acronym BAG) is published as linked data and knowledge related to buildings, public spaces, cities are captured in the BAG2 ontology (Kadaster, 2021). The building registration is reused in this project (Figure 2). Properties of the neighborhood are linked to a 'bop:Execution' and 'bop:Procedure'. As a single property can be measured (and therefore defined) through multiple methods, this structure allows for these differences to occur in the data. For example, an area's population can be measured by multiple methods, and will likely deviate across different datasets. These measurements (of type bop:Execution) are linked to the same property but describe different values measured by different procedures. Conversely, different properties (e.g. the populations of multiple areas) might apply similar measurement procedures. Lastly, each execution of a property can have one or multiple results which provide a value (Figure 2, 'Property Results'). Each result is assigned a specific time interval that denotes its temporal relevance. Thus it can be said that the result is only relevant within the time interval attributed to it. To summarize, neighborhoods of varying scales may possess multiple properties which can be measured or assessed

through multiple executions with different procedures. These executions may yield multiple results, each with a designated time interval that reflects its temporal relevance. In this project, data from multiple domains is being collected to provide a more intelligible overview for end-users. However, these datasets which describe data about the same urban areas (neighborhoods) are often stored in separate data silos with different data owners, leading to a lack of cooperation and connection. To connect these datasets, a concrete sample of the integrated data mentioned above is shown in Figure 4. This figure illustrates how data from multiple domains could be connected on multiple scale levels. Similarly, a sample of this data in Turtle format is shown in Figure 5. While Figure 4 is a visual representation, Figure 5 is a

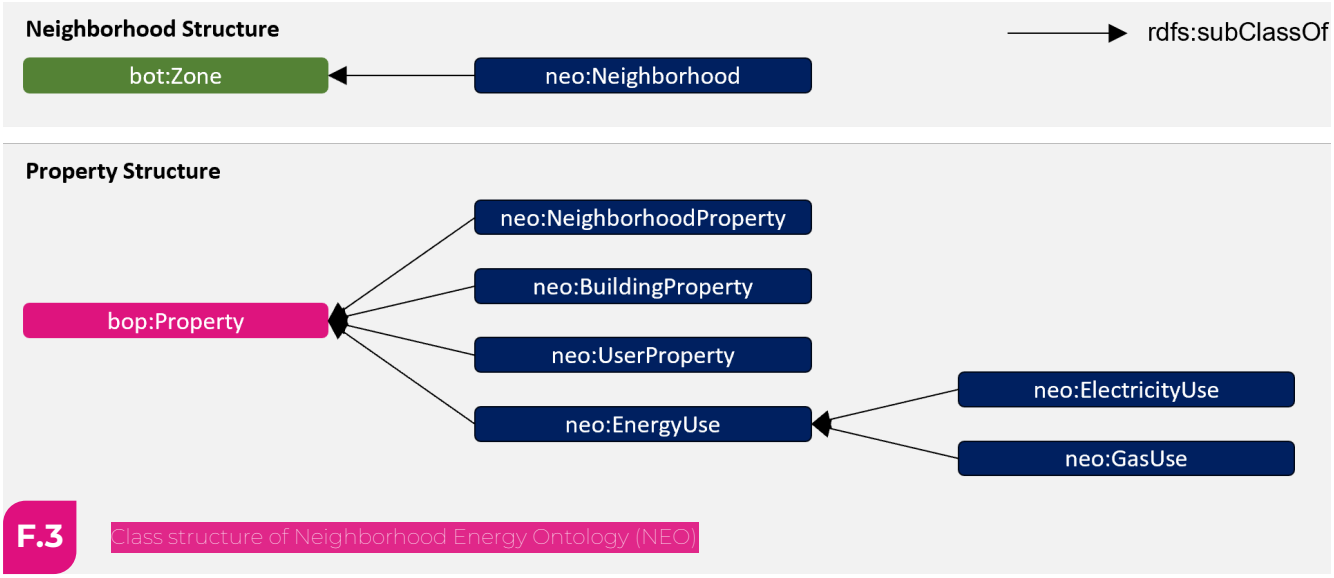
Prefix	Namespace	Color
bag2	https://bag2.basisregistraties.overheid.nl/bag/def/	Orange
bot	https://w3id.org/bot#	Green
bop	https://w3id.org/bop#	Pink
time	http://www.w3.org/2006/time#	Brown
neo	https://sanderdemeij.github.io/neo/	Blue
skos	http://www.w3.org/2004/02/skos/core#	-
rdfs	http://www.w3.org/2000/01/rdf-schema#	-
xsd	http://www.w3.org/2001/XMLSchema#	-
seas	https://w3id.org/seas/EvaluationOntology#	-

T.1 Prefixes and corresponding namespaces used in Neighborhood Energy Ontology (NEO)

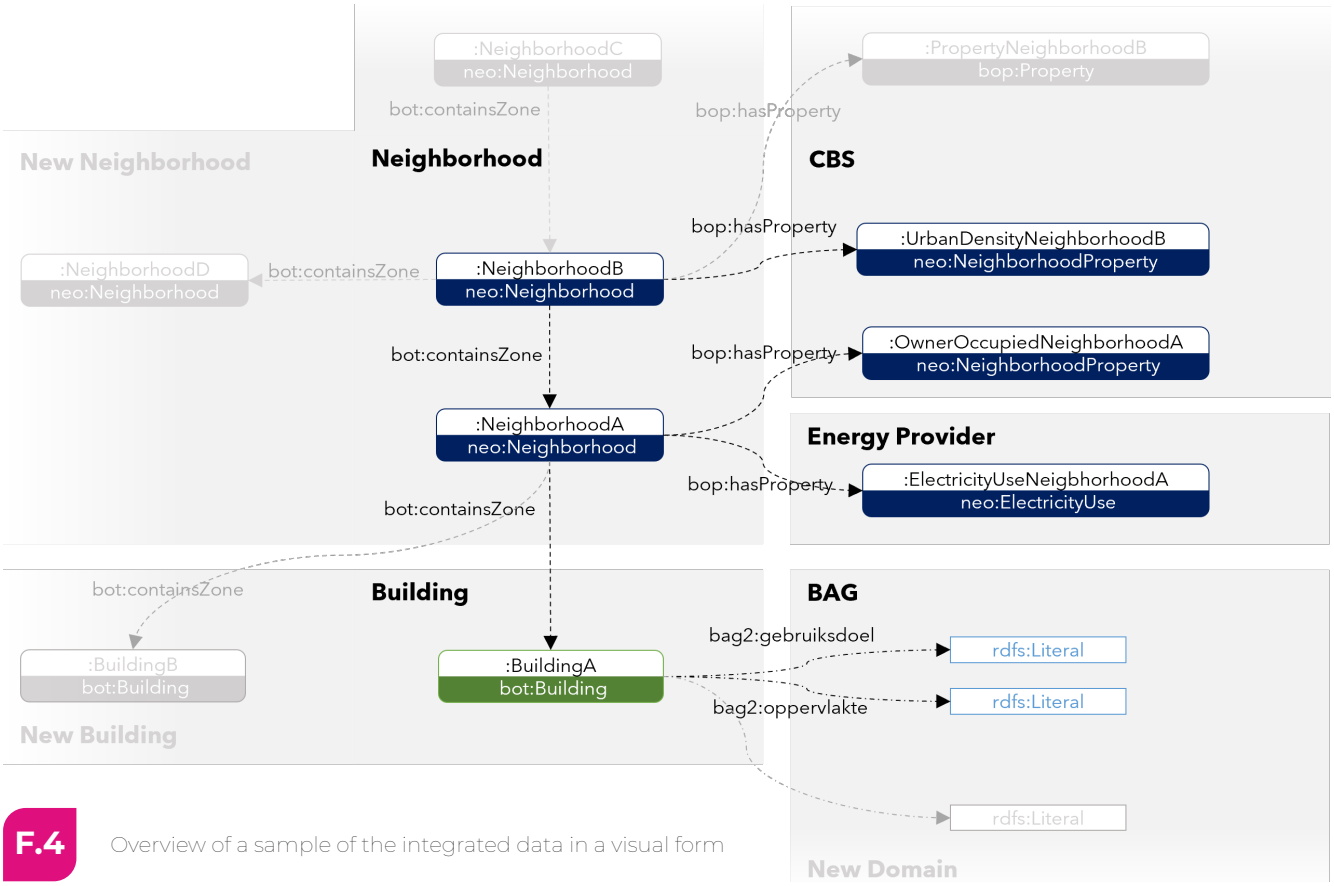


F.2 Overview of data integration method

machine-readable example of the data. By adopting this method, the data is made accessible for further analysis and interpretation by different stakeholders. Moreover, in future development, different scales and domains can be added easily (which will be compatible with the designed dashboard).



F.3 Class structure of Neighborhood Energy Ontology (NEO)



F.4 Overview of a sample of the integrated data in a visual form

```
<http://example.org/NeighborhoodB> a neo:Neighborhood,
        bop:FeatureOfInterest ;
    bag2:postcode "XXXXB"^^xsd:string ;
    bop:hasProperty <http://example.org/UrbanDensityNeighborhoodB> ;
    bot:containsZone <http://example.org/NeighborhoodA>.

<http://example.org/NeighborhoodA> a neo:Neighborhood,
        bop:FeatureOfInterest ;
    bag2:postcode "XXXXAA"^^xsd:string ;
    bop:hasProperty <http://example.org/ElectricityUseNeighborhoodA> ,
                    <http://example.org/OwnerOccupiedNeighborhoodA> ;
    bot:containsZone <https://bag2.basisregistraties.overheid.nl/bag/id/registratie/BuildingA>.

<https://bag2.basisregistraties.overheid.nl/bag/id/registratie/BuildingA> a bop:FeatureOfInterest,
        bot:Building ;
    bag2:gebruiksdoel "XXXX"^^xsd:string ;
    bag2:oppervlakte "XX"^^xsd:string .

<http://example.org/UrbanDensityNeighborhoodB> a neo:NeighborhoodProperty ;
    skos:prefLabel "urbanDensity"^^xsd:string ;
    bop:hasExecution <http://example.org/UrbanDensityNeighborhoodBProcedureA>.

<http://example.org/UrbanDensityNeighborhoodBProcedureA> neo:levelOfMeasurement "XXXX"^^xsd:string ;
    skos:definition "XXXXXXXXXXX"^^xsd:string ;
    bop:hasResult [ a bop:Result ;
            bop:hasSimpleUnit "XXXX"^^xsd:string ;
            bop:hasValue 1000.0 ;
            seas:hasTemporalContext <http://example.org/UrbanDensityTemporalContextA> ],
        [ a bop:Result ;
            bop:hasSimpleUnit "XXXX"^^xsd:string ;
            bop:hasValue 1000.0 ;
            seas:hasTemporalContext <http://example.org/UrbanDensityTemporalContextB> ] ;
    bop:usesProcedure <http://example.org/ProcedureA> .

<http://example.org/UrbanDensityTemporalContextA> a time:Interval ;
    time:hasBeginning [ time:inXSDDateTimeStamp "XXXX-XX-XXTXX:XX:XX"^^xsd:dateTimeStamp ] ;
    time:hasEnd [ time:inXSDDateTimeStamp "XXXX-XX-XXTXX:XX:XX"^^xsd:dateTimeStamp ] .

<http://example.org/ProcedureA> a bop:Procedure ;
    skos:prefLabel "XXXX"^^xsd:string;
    rdfs:comment "XXXXXXXXXXX"^^xsd:string .
```

F.5 Sample of integrated date in Turtle (.ttl) format

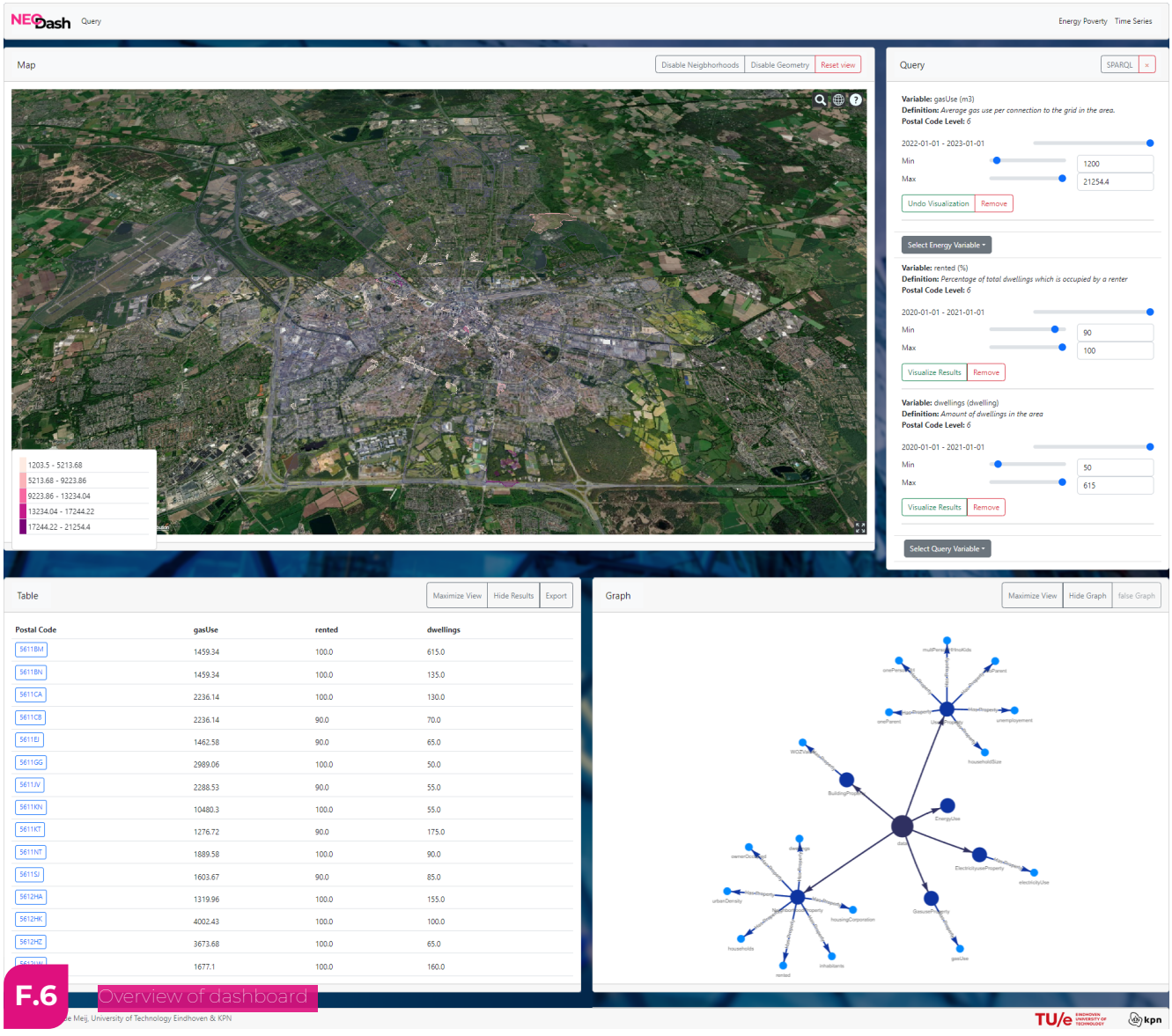
3. VISUALIZATION

To achieve the goal of more meaningful urban energy use data, a web-based viewer is created that visualizes the integrated data from multiple stakeholders. The back end of the viewer consists of two databases. First, a graph database (Ontotext GraphDB (Ontotext, 2023)) is used to store the linked data. This project incorporates data from four main sources: (1) energy use data obtained from a Dutch energy provider (Enexis Netbeheer, 2023); (2) socio-economic data collected from several sources of the Dutch Central Bureau of Statistics (CBS) (CBS, 2019); (3) energy label data procured from the Dutch Enterprise Agency (Dutch acronym: RVO) (Rijksdienst voor Ondernemend Nederland (RVO), 2022), and (4) building-related data gathered from the BAG data (Kadaster, 2021). A Python converter has been developed to convert tabular data into RDF data. The converter can integrate data from multiple datasets as long as the tabular data has a neighborhood identifier. It is a helpful addition to the dashboard described above, which can accommodate any data adhering to the data structure outlined in section 2. The aforementioned data is used as an example in this project, but the converter allows for the easy extension of new datasets in the future. Second, geometric city information is typically unsuitable for graph databases, which is why the Cesium Ion database (Cesium, 2023) is used to store and stream the geometric data. This data consists of both 2D and 3D geometries. The dashboard, a web application in JavaScript, visualizes the geometric information. This data is linked with the linked data using a Globally Unique Identifier (GUID). The web application lowers the entry barrier for endusers and can be easily built upon. The dashboard serves two main functions, visualization, and interaction (through querying). End-users can build visual queries, just like the common web shop filter bars (see Figure 6). These queries are transformed into SPARQL Protocol and RDF Query Language (SPARQL). Expert users can choose to type SPARQL queries manually. The results of these queries will be visualized in the 3D map, in a table, and a graph. The 3D map is created using the Cesium package (Cesium, 2022), and the graph is constructed using vis.js (vis.js, 2023). All these items are interactive so that if a user clicks on a certain neighborhood in the map, on a row in the table, or an element in the graph, the SPARQL query will be automatically updated and new results will pop up. Users are therefore not limited to querying functionalities but can explore the available data via different intuitive methods. These functionalities are shown in Figure 6, where the map is shown in the top left, while the query is shown in the top right. The table and graph are shown in the bottom left and right respectively. The remainder of this section will show these elements in more detail and show the different forms they can take (which correspond to different functionalities). A full overview of all functions and the

related logic and queries can be found in Appendix A, which will refer to additional appendices.

3.1 Map

As has been mentioned, the map is created using the Cesium package (Cesium, 2022), which allows for a multitude of geospatial representations. Cesium was chosen specifically to be able to visualize 3D shapes on the map. The main map displays the city under investigation (Eindhoven, the Netherlands) to the end-user, including all 6-digit postal code areas in this city. Postal codes in the Netherlands are formatted with four numbers and two letters: 0000AA. Importantly, these postal code areas can be grouped according to five- and four-digit postal codes, meaning that 0000A contains all postal codes starting with these digits. The user can zoom, pan, and rotate the view to investigate all aspects of the city. Moreover, within the map, the user can click on the postal code areas, which gives more information on the selected neighborhood. Firstly, an automatic query is generated which queries all

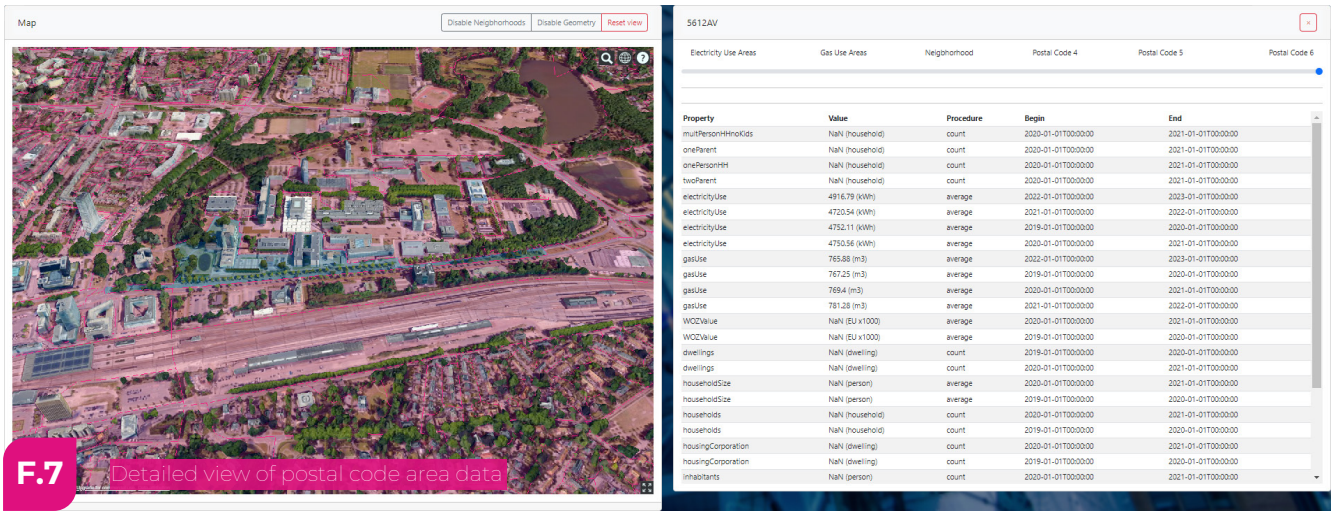


F.6 Overview of dashboard

variables associated with this postal code area. This allows the user to investigate the extent of the data available for this area, and get an initial sense of the area. The results of such a query are shown in Figure 7. For an overview of all functions, see Appendix A.1. As this Figure shows, all available data on all available spatial and temporal levels are shown. Moreover, the user can switch between different spatial measurement levels, as is shown at the top of Figure 7. In section 2 it has been explained how neighborhoods can contain other neighborhoods, and therefore neighborhoods can inherit the properties of other neighborhoods. This can result in a neighborhood inheriting properties which were measured on different spatial levels, as can be seen in Figure 8. Secondly, a combined 3D visualization of the buildings within the area is shown. This visualization consists of a Light Detection and Ranging (LiDAR) scan, combined with the 3D BAG dataset (Peters et al., 2021) and has Level of Detail (LoD) 2. As has been shown in previous figures (6, 7, 8), the dashboard shows the buildings in 3D using the LiDAR scan made available by the University of Technology Eindhoven Digital Twin Lab. This scan is available for the entire city and is visualized with differing levels of detail based on the zoom level of the map. When the user selects a specific neighborhood, the 3D volumes of the 3D BAG dataset (Figure 9) are loaded below the LiDAR layer which allows the user to click on the buildings and make the link to the BAG dataset as mentioned in section 2.

3.2 Query

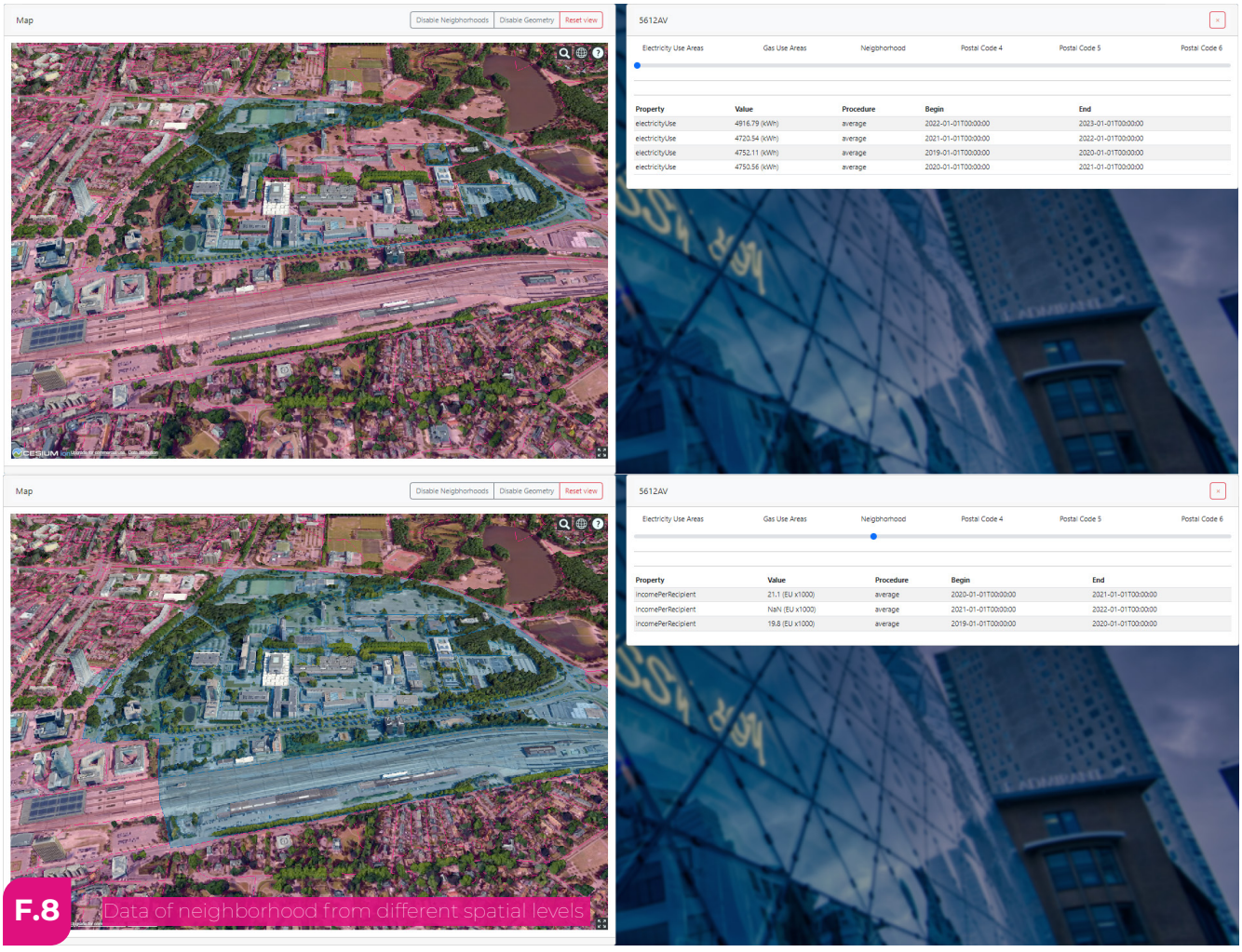
While the map can be considered the core element of the dashboard, the query functionality adds the first layer of interactive capabilities. As is shown in Figure 6, the user has the option to select energy use variables, as well as other variables, to construct a query. These variables are automatically generated based on the available instances of bop:Property in the graphs. Moreover, the user can select multiple variables to construct more complex queries and visualize the results of the query on the map. The overview of all functions is listed in Appendix A.2. The structure for constructing a query is provided in Figure 10. Firstly, the user can select one of the available properties (with different execution methods). This list of properties is created dynamically based on the available data. Secondly, the user



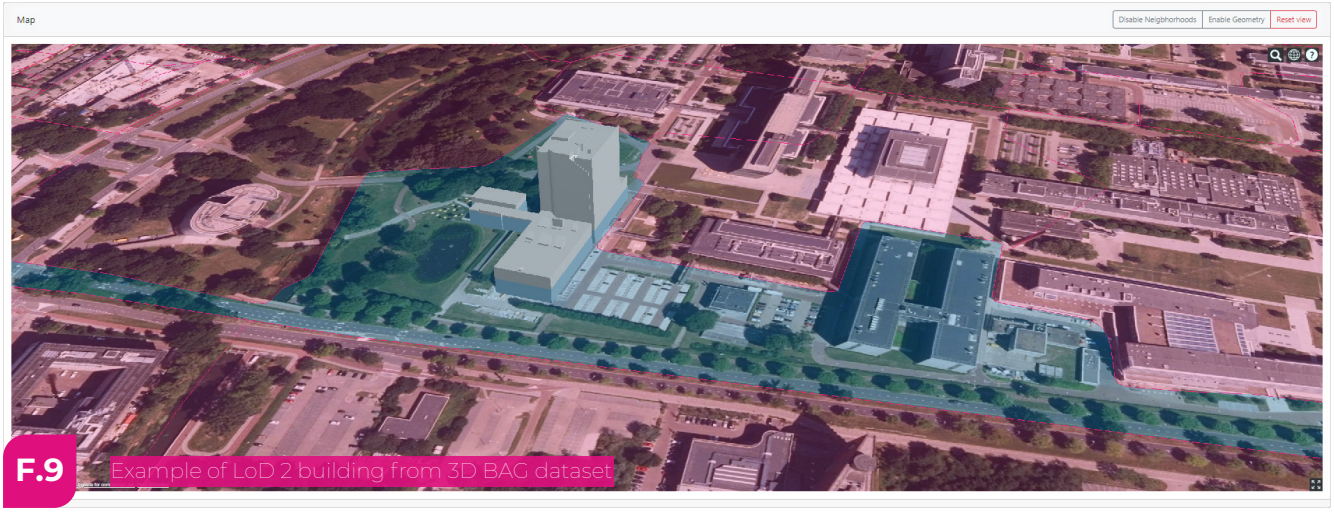
F.7 Detailed view of postal code area data

selects a property to query, after which the property is represented correctly. The representation method is based on the level of measurement indicated for the execution method (Figure 2, 'Property') and a high-level check of the found values.

Based on whether the data is numerical, categorical, or nominal, a different user interface is generated to build the visual query method. In the example provided in Figure 10, the data is numerical, and therefore the user is presented with



F.8 Data of neighborhood from different spatial levels

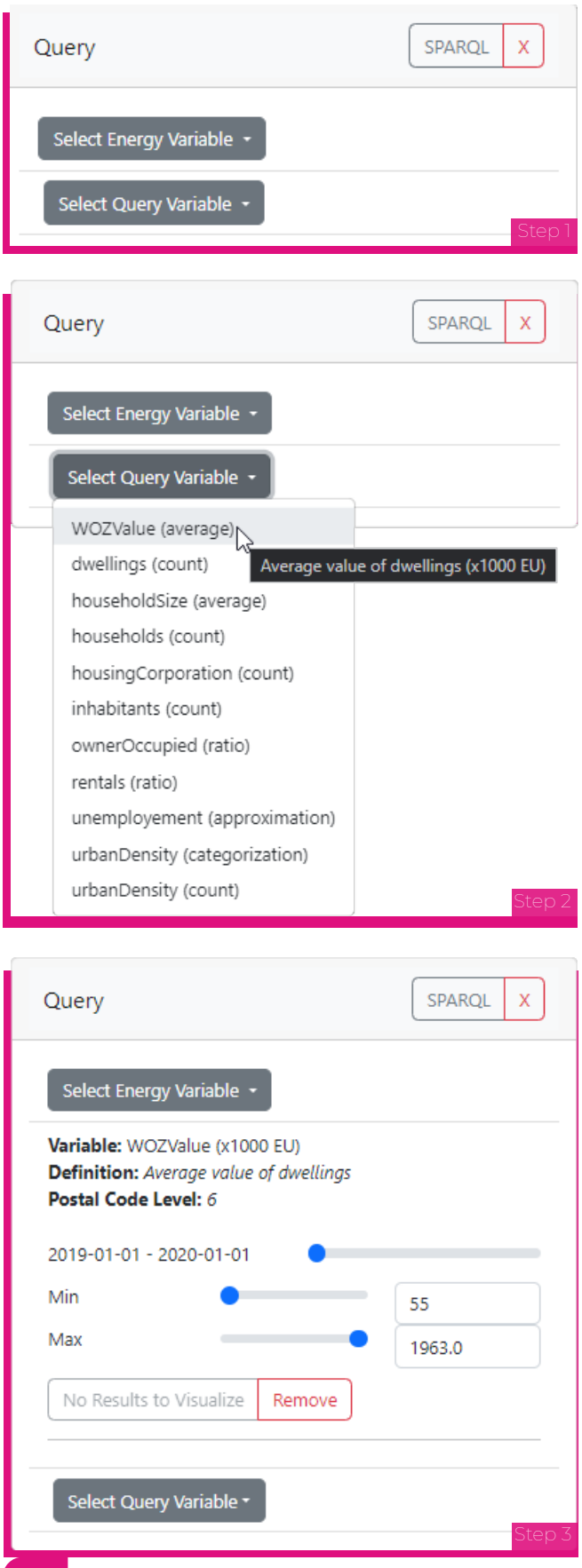


F.9 Example of LoD 2 building from 3D BAG dataset

means to query this data accordingly. Moreover, the user can select multiple properties to run a complex query. Meaning that individual SPARQL queries are constructed and executed, which all return the correct postal codes and their values. When multiple variables are queried the overlap between the found postal codes is established, and these postal codes are shown on the map. When the user chooses to visualize the results, the found values for that variable are converted to five bins, each represented by a color (as shown in Figure 6). This allows a more intuitive way to explore the spread of the found values over the city. The user can switch between the visualization of all the found variables, however, the visualization of multiple variables at once is not available as the interpretation of this visualization would be too complex.

3.3 Table

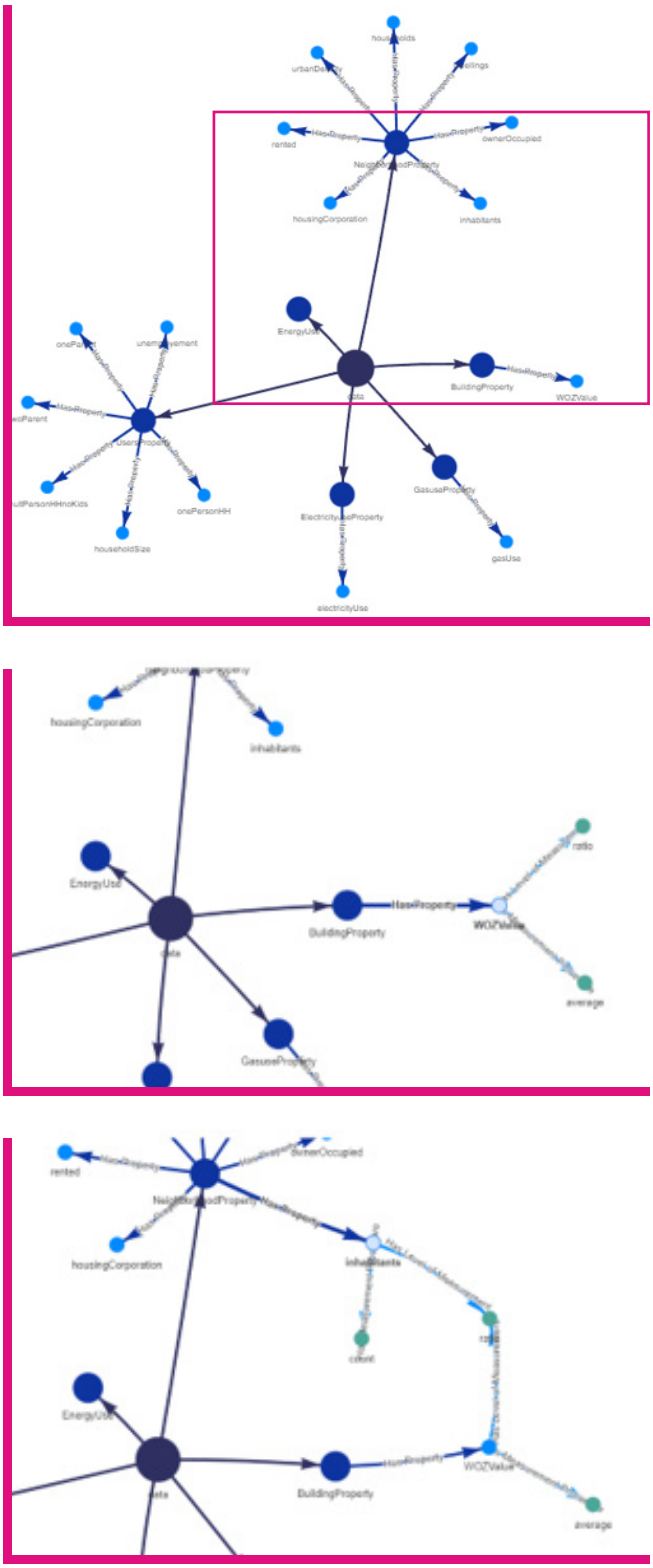
While the main purpose of this project is to make urban (energy) data more meaningful by visual means, a more traditional data representation is also included. As Figure 6 shows, a table of the results found by the query is created where each row represents a postal code that is returned by the query, while each column represents the corresponding value of the variables included in the query. The user can sort the table by any of the variables from low to high or the reverse. Moreover, the user can select a postal code to investigate in more detail by clicking on it in the table. The map will update to show more details of this neighborhood as has been explained in the previous section. An overview of all functions is shown in Appendix A.3.



F.10 Process of constructing a query using the dashboard

3.4 Graph

Lastly, the graph visualizes the structure of the data to the user. Here, the user can investigate the data on a higher level and discover what data is available and create new queries. Based on the available data, the graph shows the connections of the properties, firstly based on their super-property class (as shown in Figure 3). When the user selects a property, the level of measurement, unit, and measurement procedure are shown (Figure 11, top). Moreover, when any of these aspects are selected their relationship to other properties is also visualized (Figure 11, middle), allowing the end-user to investigate the nature of the data in more detail. Using these functionalities and their interactions, the user can query data that crosses multiple domains using one method. The user can discover new aspects of the data which were previously impractical to discover and therefore gain new insights about their individual questions. This dashboard could be used this way to answer existing questions or formulate new questions which the user was unable to form before. All functions of the graph are listed in Appendix A.4.



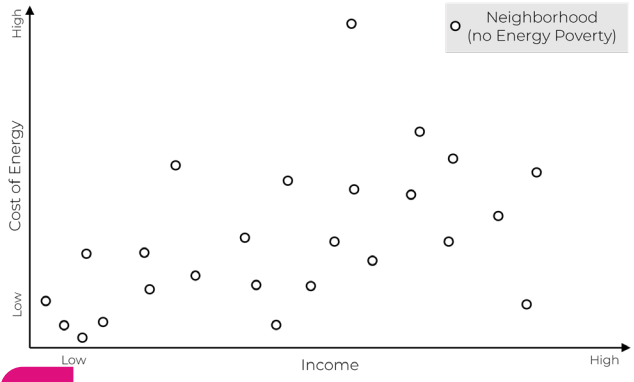
F.11 zInteraction with graph representation of data

4. USE CASES

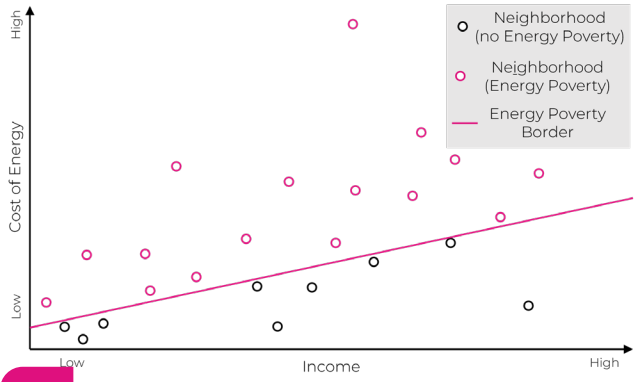
The previous sections have shown how NEO can be implemented in the Neo Dash semantic digital twin. While this digital twin is a first step towards more meaningful urban data, it does not necessarily solve the challenges described in section 1. Therefore, this section will go into detail on several, related, use cases which will show how this digital twin and the proposed semantic data structure can make urban data more meaningful. Firstly, section 4.1 will show an analysis of the concept of energy poverty. This analysis will show how data from disparate data sources, temporal resolution, and spatial scales can be combined to gain more insight into a relevant societal issue. Secondly, section 4.2 will be explored how the proposed digital twin can also be implemented on a very high spatial and temporal resolution by exploring the possibilities of real-time data implementation. Lastly, section 4.3 will explore the possibilities to use the data structure and dashboard to perform predictive analytics and enrich the analysis done in section 4.1.

4.1 Energy Poverty

The first use case aims to investigate the societal issue of energy poverty (or, fuel poverty). However, before going into detail on the added value of the created dashboard and data structure, it is necessary to provide a definition (or several definitions) of the concept of 'energy poverty'. 'The definition of fuel poverty is important for policy formulation; for determining the scale and nature of the problem; targeting a strategy and monitoring progress' (Moore, 2012). According to Moore (2012), an early definition of energy poverty was an expenditure on energy service exceeding 10% of a household's income. However, this definition was provided in 1991 and since then several alterations have been made, moreover, this definition was given in the UK context. A more recent definition is 'Low Income, High Cost' (Dutch: Laag Inkomen, Hoge Energierekening (LIHE)) (Department for Business, 2020; Mulder et al., 2023). Following the definition provided by Mulder et al. (2023), this definition considers households with low income and a high energy bill, where low income is considered to be within 130% of the poverty line and a high energy bill is an energy bill above the median (for the year 2019). An even more recent and expansive definition is 'Low Income, Low Energy Efficiency' (Dutch: Laag Inkomen, Lage Energetische Kwaliteit (LILEK)) (Department for Energy Security & Net Zero, 2023; Mulder et al., 2023). This definition has the same definition for low income as the LIHE definition, however, a dwelling is considered to have low energy efficiency when the expected energy use is lower than an average C-label dwelling. As can be seen from these three definitions, several concepts are combined to evaluate energy poverty. These concepts range from socio-demographic variables (income) to building variables

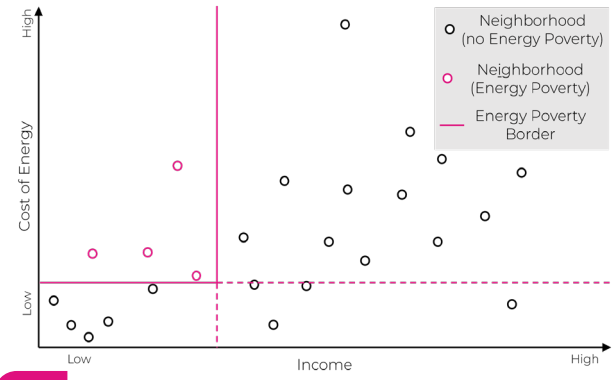


F.12 Conceptual representation of energy poverty visualization

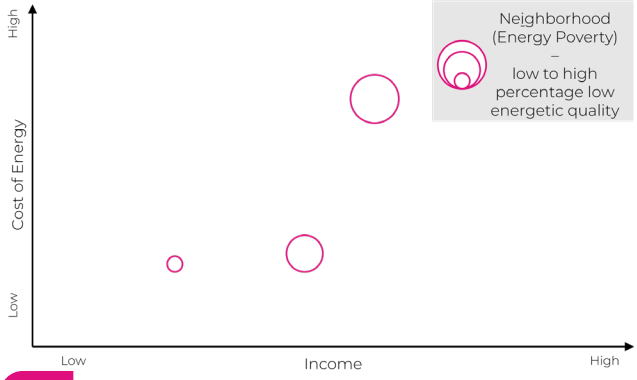


F.13 Conceptual representation of '10% of Income' definition of energy poverty

(energy label), which indicates that data from several data sources need to be combined to easily assess energy poverty on an urban scale. Therefore, the goal of this use case is to evaluate these three definitions of energy poverty, and thus show the potential of semantic web technologies and associated tools. To achieve this goal, a visual approach is taken in the dashboard design where the income and energy costs are shown per neighborhood (Figure 12). In this figure, each dot represents a neighborhood, which has a corresponding income (average per neighborhood from 2021) and cost of energy (average energy use per connection from 2022, combined with an average energy cost of 2022). Considering the first definition discussed above, this would mean that a line can be drawn where the cost of energy exceeds 10% of the annual income in the neighborhood (Figure 13). In this figure, the neighborhoods which fall below this line can be considered to not be at risk of energy poverty, conversely, neighborhoods above this line can be considered to be at risk of energy poverty according to this definition. Similarly, according to the LIHE definition, two lines can be drawn: 130% of the poverty line and the median energy cost (Figure 14). In this figure, the neighborhoods in the upper left quadrant can be considered to be at risk of energy poverty. Considering that the vertical line represents the demarcation of 130% of the poverty line, and thus neighborhoods falling to the left of this line are below this benchmark. Similarly, the horizontal line represents the median energy cost, and thus neighborhoods that are above this line have higher energy costs than this benchmark. Considering the final definition, a slightly different conceptual approach is taken. The previous definitions have considered energy poverty on a neighborhood scale, however, as the final definition includes building-level data (energy labels) such a spatial scale becomes problematic. Therefore, only the neighborhoods which are considered at risk according to the LIHE definition are considered in the final analysis. Within this set of neighborhoods, if a building is present within that neighborhood that has an energy label below C, this neighborhood is shown in the visualization (Figure 15). As will be explained in more detail below, the radius of the dot represents the relative amount of dwellings at risk of energy poverty. Implementing this conceptual visualization gives the following dashboard features. The data used in this use case is summarized in Table 2. In this table, the variable is named, together with its spatial level. 'PC6' indicates a postal code level with

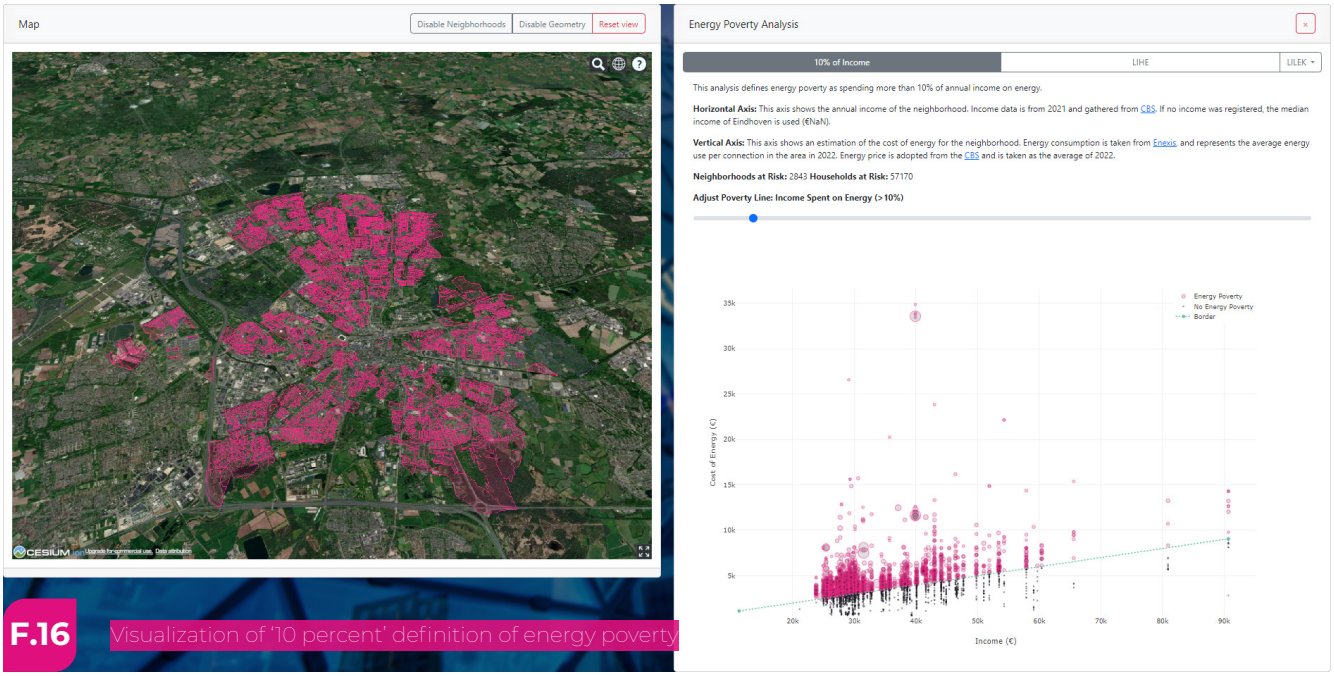


F.14 Conceptual representation of LIHE definition of energy poverty



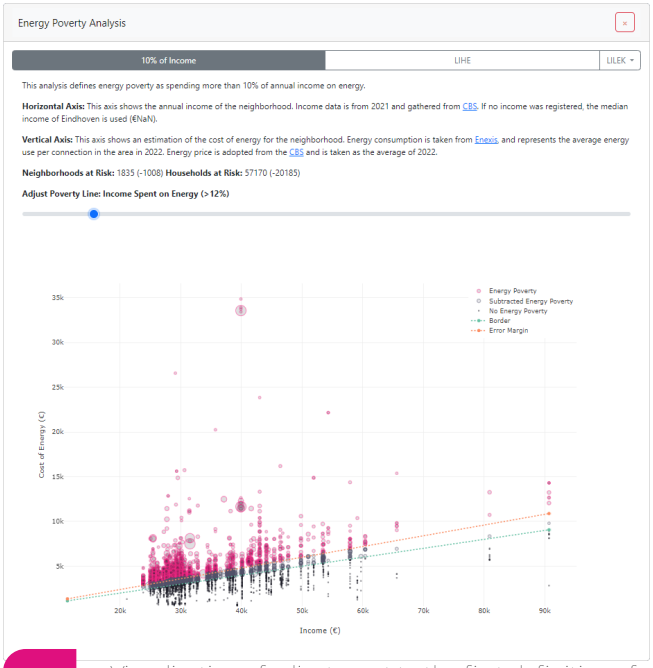
F.15 Conceptual representation of LILEK definition of energy poverty

6 digits (0000AA), which is the highest level of urban detail in the Netherlands. 'Buurt' indicates a spatial level created by the Central Bureau of Statistics and is a collection of 'PC6' neighborhoods. First of all, the concept of Figure 13 is actualized, as is shown in Figure 16. This figure shows several aspects of this functionality of the dashboard. Firstly, the top part of the 'Energy Poverty Analysis' box shows that the user can switch between the three definitions. Moreover, the definition and data sources are shown in order to inform the user. Secondly, the actual graph is visualized where the pink dots represent neighborhoods at risk of energy poverty, while the grey dots represent neighborhoods that are not at risk (according to this definition). These colors correspond to the map visualization (section 3.1), which is updated dynamically based on the graph. Moreover, it might occur that income (Table 2 - 4) is unknown for a certain neighborhood, in this case, the income is imputed using the mean of the available data. These neighborhoods are visualized separately to avoid confusion in further analysis by the user. Lastly, the number of neighborhoods at risk is summed to give the user an overview of the magnitude of the problem at hand (which can also



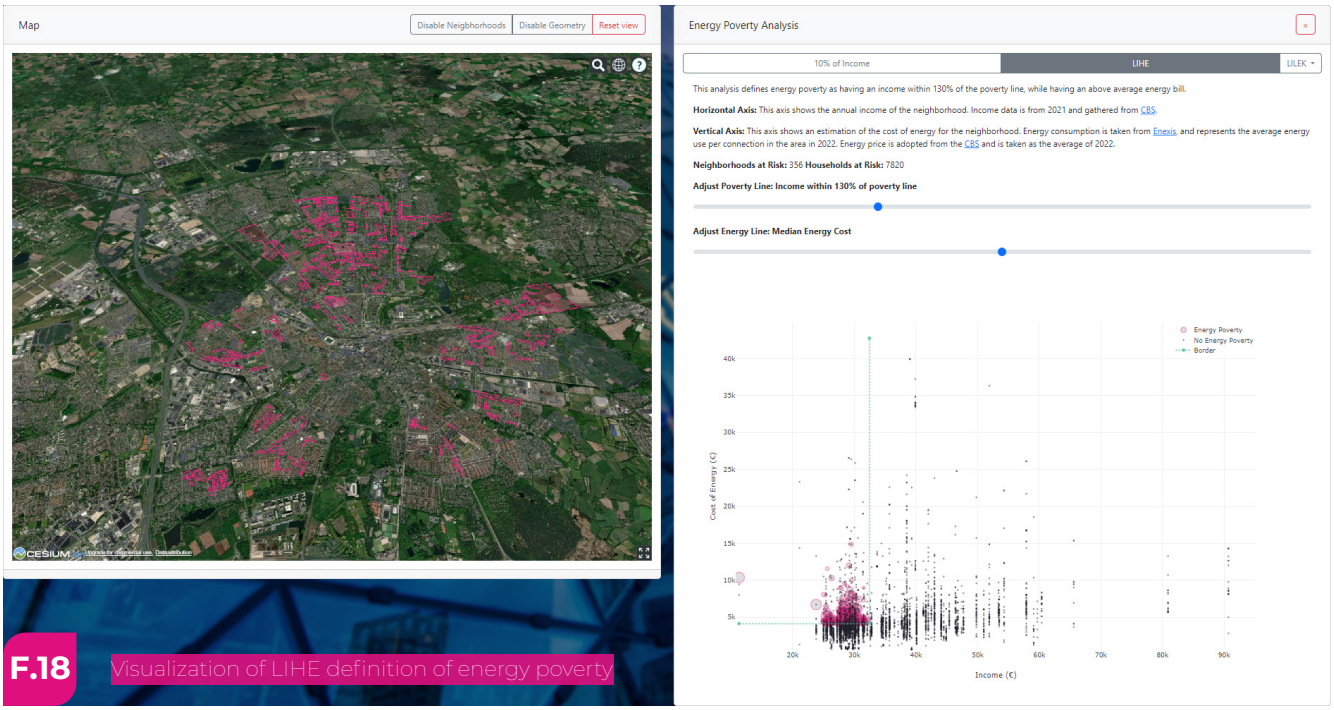
F.16 Visualization of '10 percent' definition of energy poverty

be compared between definitions). As the total amount of households per neighborhood is known (Table 2 - 5), the amount of households at risk is also shown. As this graph shows, there are several neighborhoods that are close to the border, which might be of interest to the user. By slightly altering the given definition, the user might investigate how sensitive certain neighborhoods are to small deviations in income spent on energy services or costs of energy services. This analysis is made possible by the slider shown in Figure 16, which adjusts the border as is shown in Figure 17. As this figure shows, by adjusting the border to 12% of income, more than 600 neighborhoods are now no longer at risk of energy poverty according to the data. This difference is shown in the graph by the gray dots (which are also represented in the map visualization). Similarly, if the border would be adjusted in the other direction, additional neighborhoods would be considered at risk of energy poverty. Considering the second definition of energy poverty, a similar visualization is implemented, as shown in



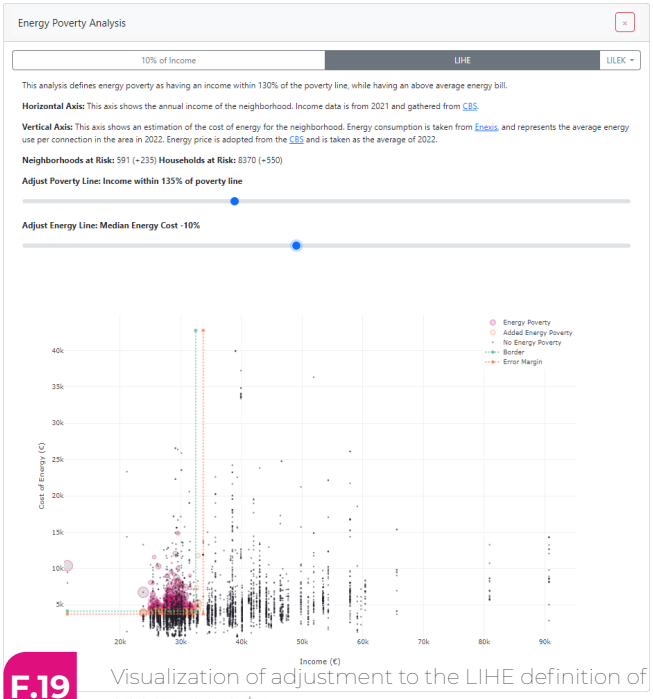
F.17 Visualization of adjustment to the first definition of energy poverty

Figure 18. The concept of this visualization has been explained according to Figure 14, however, the actual implementation is more complex considering that different levels of poverty are defined by the Dutch government based on household composition (Centraal Bureau voor de Statistiek, 2022). As the data on



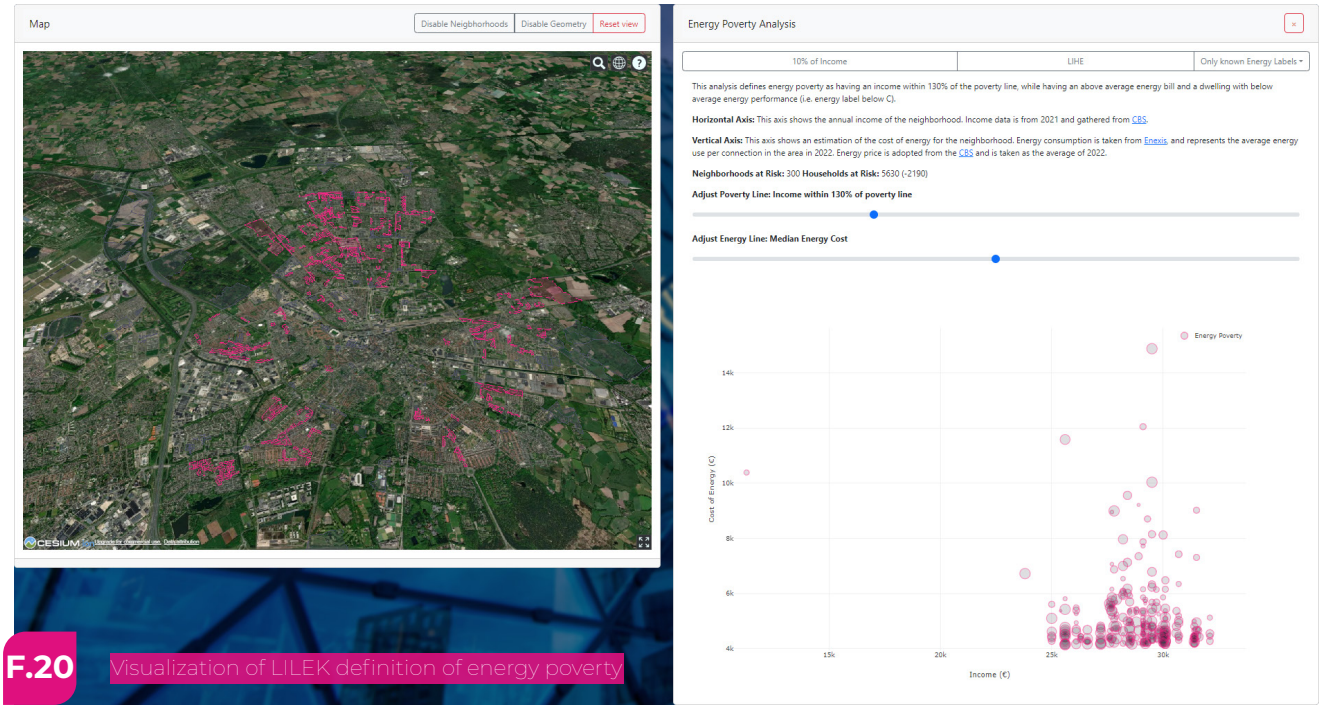
F.18 Visualization of LIHE definition of energy poverty

household composition (Table 2 - 6, 7, 8 & 9) is available, it is calculated how many households are considered energypoor given the different levels of poverty, and the available energy cost data. If no households are considered energy poor according to the LIHE definition, the neighborhood is visualized as not at risk. The borders shown in Figure 18 are the least stringent (highest poverty line) definitions, which might result in neighborhoods falling within the borders of the graph representing energy poverty, however, being marked as not at risk of energy poverty. For the previous definition, it has been described how the definition of energy poverty can be adjusted to explore the sensitivity to change. A similar functionality is available for the LIHE definition, however, in this graph, two dimensions need to be altered as this definition is dependent on two factors (income and median energy cost). This results in a graph similar to the graph shown in Figure 19, where the orange dots represent neighborhoods that are newly marked as at risk of energy poverty according to the adjusted definition of

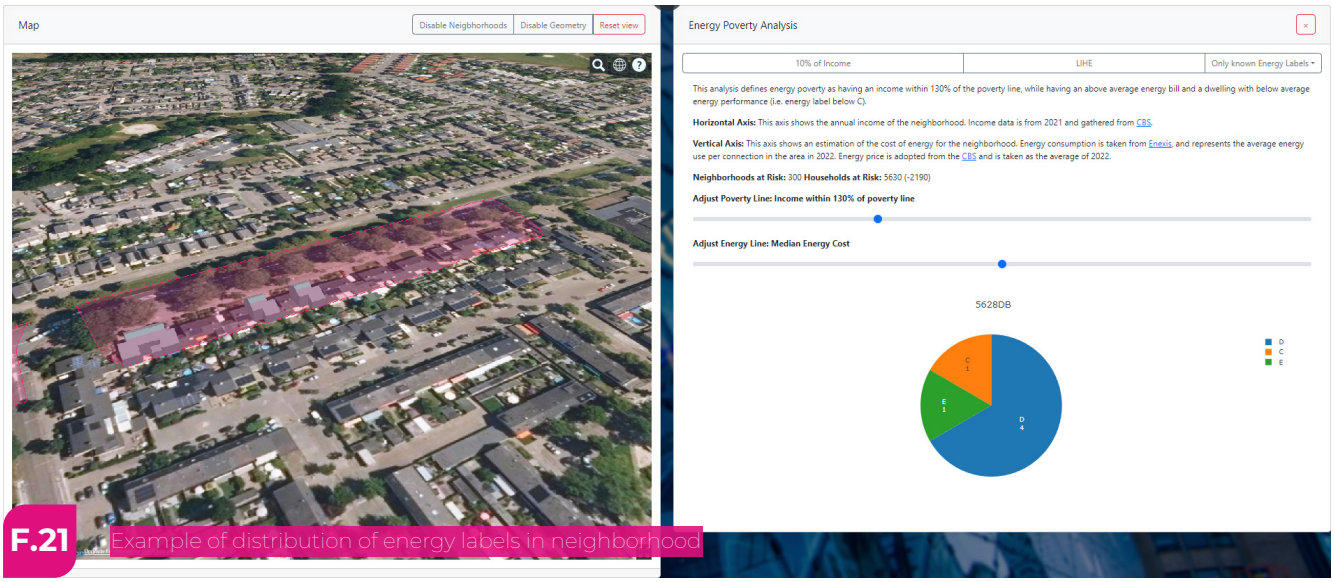


F.19 Visualization of adjustment to the LIHE definition of energy poverty

LIHE. Regarding the last definition, the concept of Figure 15 is translated into the visualization shown in Figure 20. As has been mentioned before, the neighborhoods represented in this graph are considered at risk of energy poverty according to the LIHE definition, while also containing a dwelling with an energy label worse than C. The radius of the dots



F.20 Visualization of LILEK definition of energy poverty



F.21 Example of distribution of energy labels in neighborhood

is determined by the relative amount of such dwellings in the area, which means that larger dots represent neighborhoods with relatively more dwellings at risk of energy poverty. To provide more insight into individual neighborhoods, the user can select a neighborhood which is shown in Figure 21. This graph shows the actual distribution of energy labels in the neighborhood. Moreover, if the user selects a label in the graph, the corresponding dwellings are shown on the map (see Figure 21, energy label D is selected). From a user perspective, this analysis shows how different definitions give different insights into the societal issue of energy poverty. This dashboard enables the user to switch between definitions and argue for certain policy measures based on these different definitions. Moreover, this use-case shows how semantic web technologies provide benefits for complex questions which require information from different sources. The LILEK analysis combines energy use, income, household composition, number of households, and energy label data. Without using Linked Data such analysis would be hard to conduct (repeatedly). Using the proposed ontological structure, and

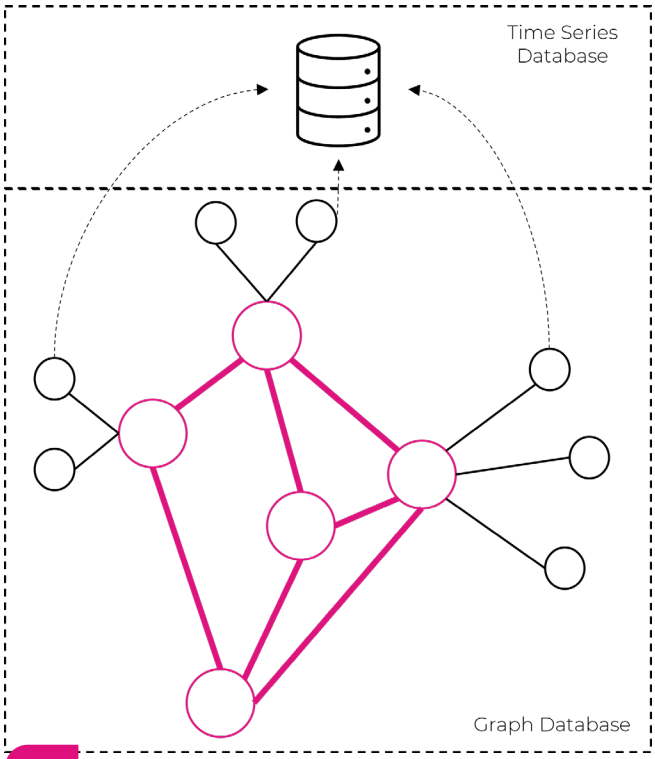
Variable Name	Spatial Level	Unit
1. Electricity Use	pc6	kWh/ connection
2. Gas Use	pc6	m3/connection
3. Dwelling Value	pc6	x1000 EU
4. Income	Buurt	x1000 EU
5. Households	pc6	-
6. Multi person household (no children)	pc6	-
7. One Parent Household	pc6	-
8. One Person Household	pc6	-
9. Two Parent Household	pc6	-
10. Energy Label	Building	A++++ - G
11. Built Year	Building	year
12. Area	Building	m2

T.2 Overview of data used in Energy Poverty use case

the designed dashboard, the user can explore the data quickly and repeatedly. In addition, the analysis is made more meaningful through the ability to explore the problem visually. The results of the analysis are visualized geographically as well as in a graph, allowing the end-user to localize interesting areas of the city. It is suggested that this adds additional meaning to the analysis, as the user is not only able to explore the results in a graph, but also geographically.

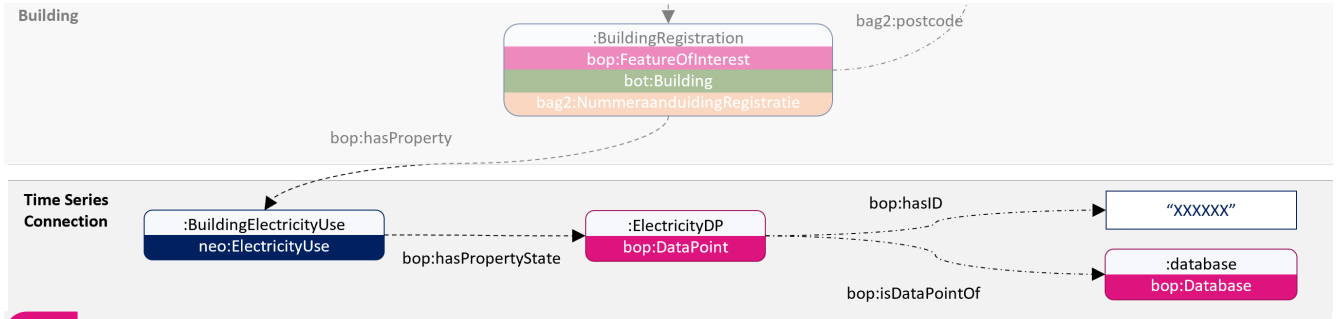
4.2 Time Series Data

Generally, graph databases (as used in this project) can be considered unsuitable for time series storage. A high-level explanation for this characteristic of graph databases is the fact that for every time-series data point, a new set of triples (see Figure 1) needs to be created, resulting in a quickly exploding graph structure when time-series data is recorded with a high frequency.



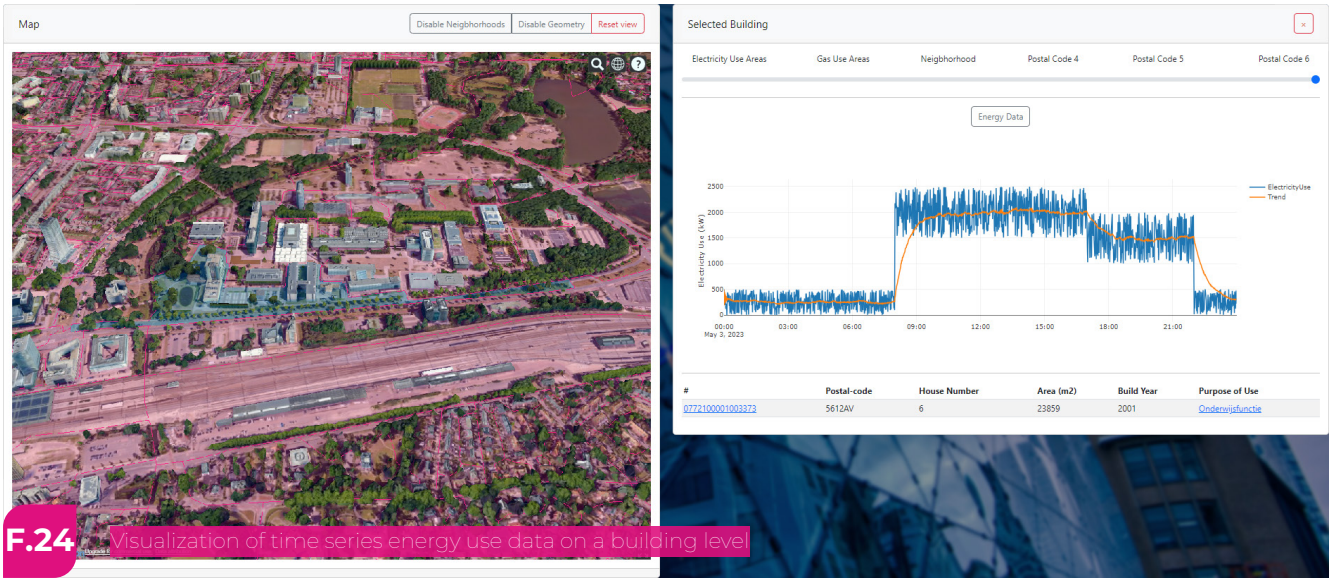
F.22 Conceptual explanation of the connection between Graph- and Time Series databases

Therefore, time-series data is stored in databases that are created specifically for those purposes. The previous use case has shown how mostly aggregated data is used (aggregated on a spatial as well as a temporal level). This type of data is suitable for certain purposes, however, also introduces uncertainties. The energy use variable discussed in the previous use case shows these uncertainties quite clearly. This data is the yearly average energy use of all the connections to the energy grid in that neighborhood. Aggregating data over a year eliminates the fluctuations over seasons and days. Moreover, energy use might fluctuate greatly between buildings within a neighborhood. Therefore, this use case will explore the challenge of integrating the proposed data structure with time series data. While this type of integration might not be required in larger urban analysis, it can provide additional insight on an individual building level. This will make urban data more meaningful to individuals or small scale urban end-users, such as real-estate managers. Moreover, this will show the transitive nature of NEO, where data can be defined on multiple scales (and queried on those differing scales). Here, the main method used is to use the graph as a 'map' which can guide to the right database to query for the time series data. This concept is explained in Figure 22. This figure shows an example of a graph consisting of



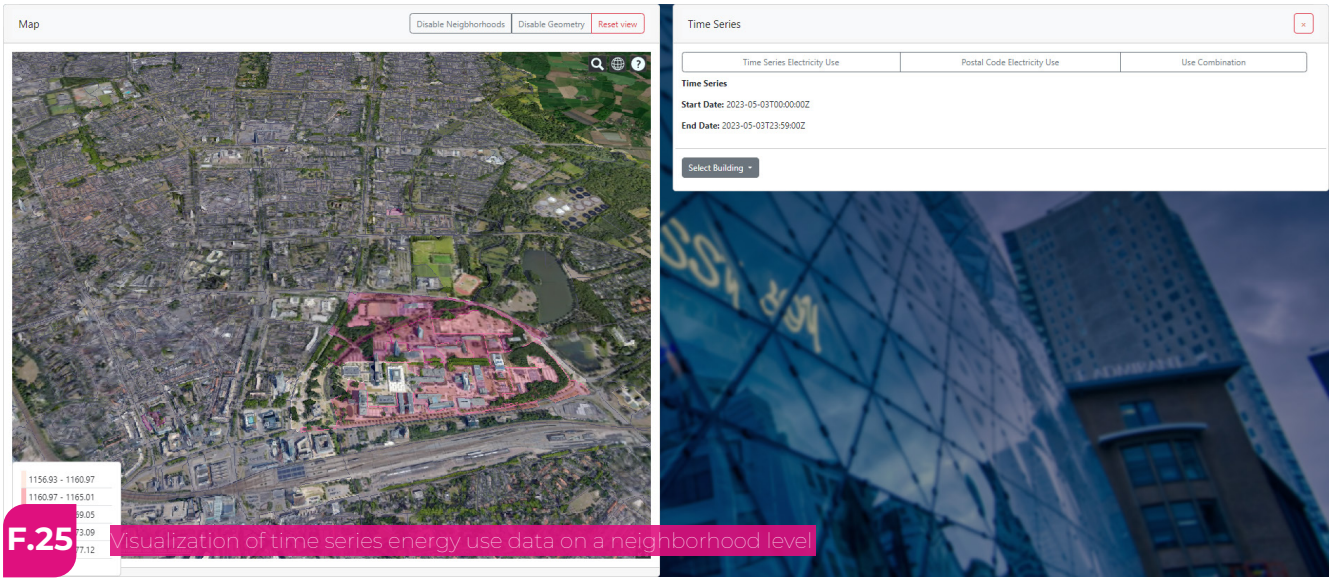
F.23 Overview of the integration of Graph- and Time Series data

edges and nodes (triples in this project). Some of the black nodes have a related data element which is stored in an external times series database. Such a structure would allow for a query where the graph can be explored (like a map) to search for nodes that have a related time series data point. As has been discussed, in order to overcome some of the uncertainties of the previously used data, higher resolution energy use data will be explored. This data will consist of time series data on a building level, as formally shown in Figure 23. This Figure shows how the available structure of BOP (https: //w3id.org/bop#) is implemented in this project. A building can therefore have a certain property (in this use case electricity use) which has a property state pointing to a data point. This data point is part of a particular (time series) database and has an associated ID. This ID will be used to query the correct data for the associated database. In order to show the potential of this structure, a day of energy use is generated randomly for a set of buildings on the University of Technology Eindhoven campus. Randomly generated data is used as such type of data is not available due to privacy reasons, however, the goal of this use case is to explore the added value of making such data available (and exploring the challenges). An example of such data and its implementation into the dashboard is shown in Figure 24. Here, the dashboard queries the Graph database with the URI of the selected building and its corresponding data point(s) if available. If such a data point is available, the corresponding ID and database identification are used to retrieve the available time series data. Using this data on an urban scale results in the dashboard as shown in Figure 25. Here, a similar query is constructed as described before, however, for all buildings which have time series data available. An average of the available data on a neighborhood level is then used for visualization, as shown in Figure 25. However, as can be seen, only data is available for the campus, therefore, it might be desirable to combine both time series and yearly average data, using time series where possible and the yearly average otherwise. This implementation is shown in Figure 26. At first glance, no difference is visible between this implementation and the use case discussed in Section 4.1. However, investigating the time frames shown

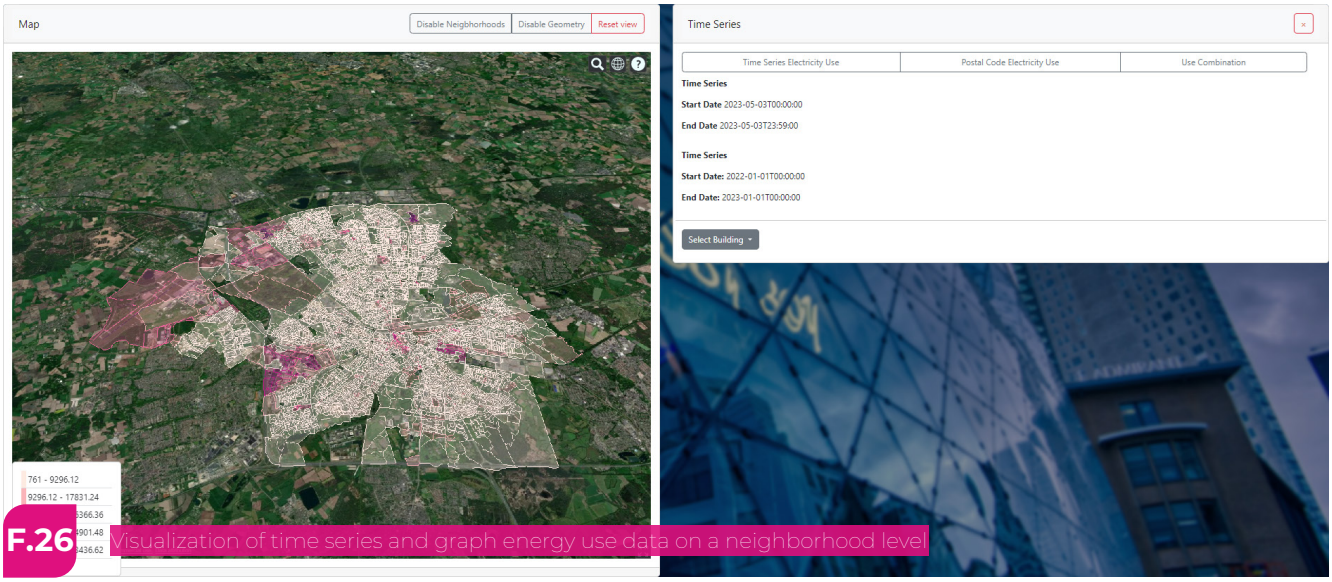


F.24 Visualization of time series energy use data on a building level

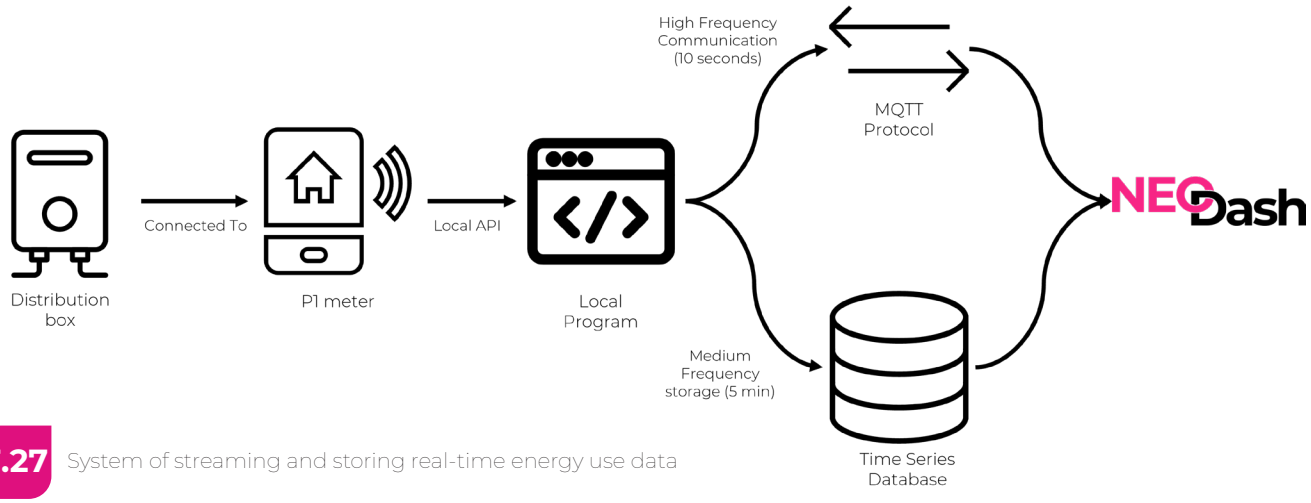
4. Use Cases - Time Series Data



F.25 Visualization of time series energy use data on a neighborhood level

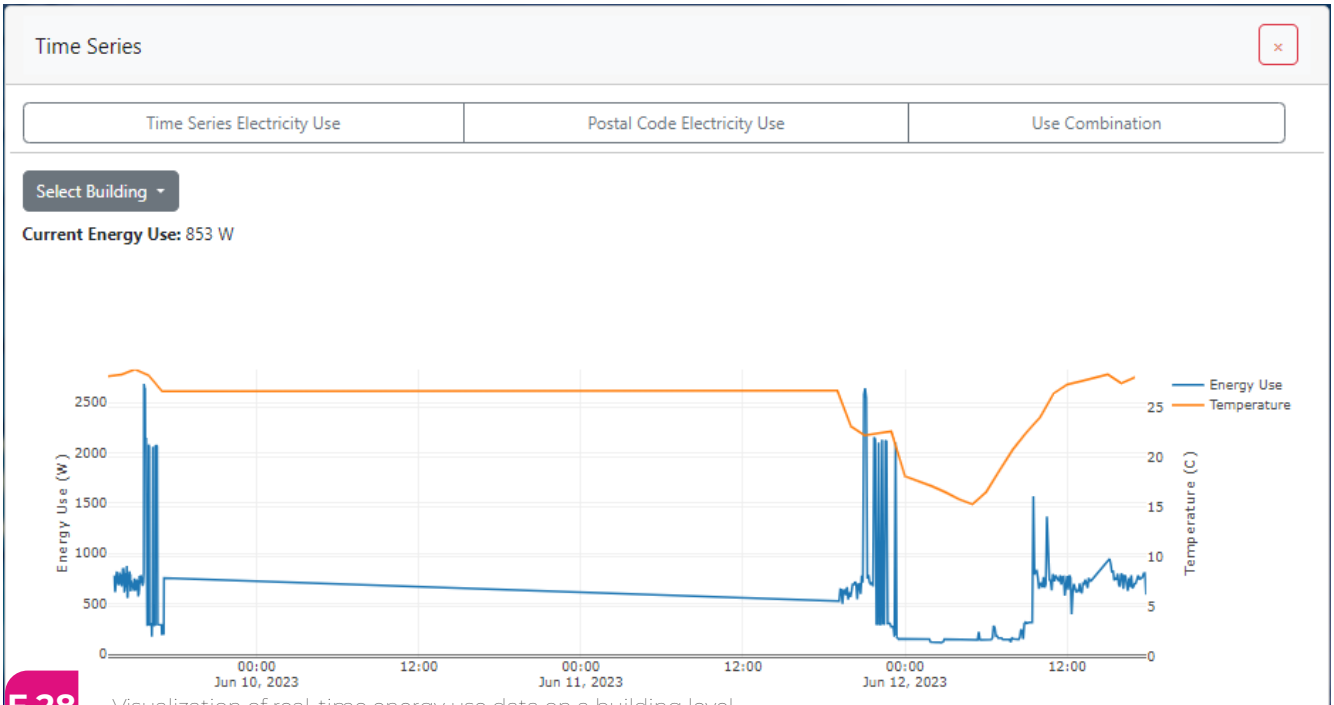


F.26 Visualization of time series and graph energy use data on a neighborhood level



F.27 System of streaming and storing real-time energy use data

in Figure 26, some of the challenges of this use case become clear. The yearly average energy use data spans an entire year, while the time series average only spans one day, which might limit the usefulness of such an implementation. However, this use case shows how this data can be useful to the building man agers of this particular urban area (university campus). They can investigate energy use for all sections of the campus, while also having access to individual buildings' energy use. Therefore, they can move through different spatial levels, to gain more insight. Moreover, they could link urban data of the campus (demographics, greenery, etc.) to gain even more insight. However, high-resolution energy use data is not currently available, therefore, a second implementation is explored where energy use of a dwelling is used in real-time. This implementation is created in order to show the future potential of the dashboard and the value of the incorporation of graph and time series databases. In order to store and stream this data, a P1 meter (HomeWizard BV, 2023) is connected to a dwelling in Eindhoven. This meter is accessible through a local API, which is used in a continually run program to store the data every 5 minutes while streaming the data every 10 seconds (see Figure 27). In order to continually stream the data, the MQ Telemetry Transport (MQTT) protocol is used. This protocol uses a central broker, where data can be published on certain topics. Once subscribed to these topics, the streamed data can be retrieved. The central broker and publishing topic are encoded in the graph structure shown in Figure 23, where the database indicates the correct broker, while the ID provides the correct topic. The topic is dependent on the type of data that is streamed (in this case electricity use), the city, and the building in that city, resulting in topics of the following format: 'CITY/VARIABLE/BUILDING URI'. The implementation of this system is shown in Figure 28. The dashboard queries for the correct broker and topic for the building and shows to most recently published data. While



F.28 Visualization of real-time energy use data on a building level

also querying a time series database as described before, to show all the available data. This implementation shows how it is possible to stream and store energy use data for buildings and use that data on an urban level. Moreover, as Figure 28 shows, this type of data can be matched to external data to gain new insights as also the temperature of the recorded time is shown using an external API. As an example, Figure 28 shows certain peaks in energy consumption which can be related to the relatively high temperature on the recorded day. The building owner, or real-estate manager, can now investigate these trends and relate them to additional building and urban characteristics. Concluding, this use case has shown how the proposed ontological structure can be used and extended to incorporate time series data and gain more insight into urban energy use. While the use case described in section 4.1 explored energy use on a larger urban scale, this use case has shown how the semantic digital twin can be implemented on a smaller urban scale to make energy use more meaningful on an individual building or small neighborhood level.

4.3 Predictive Analytics

The previous two use cases have focussed on the analysis of existing data and visualizing that data in order to gain better insights. However, users might also require some predictive analytics as part of the dashboard. Therefore, this section will delve into a possible implementation of such an analysis. Section 4.1 discussed how the energy labels of individual dwellings are used, however, not all buildings in the city under investigation have known energy labels. The data includes 136122 buildings labeled as a dwelling (Dutch: 'woonfunctie') or with an unknown function. Of these buildings, 51099 (37.5%) have an unknown energy label. Therefore, the goal of this use case is to categorize these buildings according to the Dutch energy label system based on the available data. The data used for this analysis is a combination of data from the Cadastre, Land Registry and Mapping Agency of the Netherlands (Dutch: Kadaster), Central Bureau of Statistics, and Energy Provider. More concretely, on a building level the construction year and footprint (m2) are used, while on a neighborhood level average building value (Dutch: 'WOZ waarde'), electricity, and gas use are used. In this analysis building and neighborhood-level data are combined, where it is assumed that the average data from the neighborhood is applicable to the buildings in that neighborhood. As an example, if a building is built in 1970 and is 100 m2, while being situated in a neighborhood where the average building value is €200.000, it is assumed that that building is worth the same amount (the value of all buildings in that neighborhood is assumed to be that amount). The SPARQL query to gather the previously mentioned data is shown in Figure 29 . In order to eliminate neighborhoods where energy use is exceptionally high due to industrial activity, only neighborhoods are included where the percentage of dwellings (and buildings with a non-defined purpose of use) is above 95%. Moreover, data is gathered from 2020, which is the most recent year all previously mentioned variables are available. Considering the buildings with an unknown energy label, it might be most advantageous to label them with a new energy label ranging from A++++ to G. However, initial exploration of prediction models showed that this yields very low accuracy. Therefore, another solution has been

found. Regarding the LILEK definition of energy poverty discussed in Section 4.1, it is sufficient to know if a building has an energy label better or worse than C. This simplifies the categorization task as only two categories remain. The process for training and using the model is shown in Figure 30. For the categorization model, a Random Forest (RF) model is used. Comparing several models (logistic regression, Tree algorithm, Random Forest, Neural Network) the RF showed to best performance relative to processing time. Moreover, while this use case is aimed at showing the potential of predictive analytics within the proposed dashboard and data structure, the process of incorporating such an analysis is the main goal, not the actual performance of the model. The model is trained using a 75/25 train-test split, where the data is normalized (between 0 and 1) in order to eliminate outsized effects of the larger

		Predicted	
		<C	>=C
Actual	<C	7373	677
	>=C	626	3146

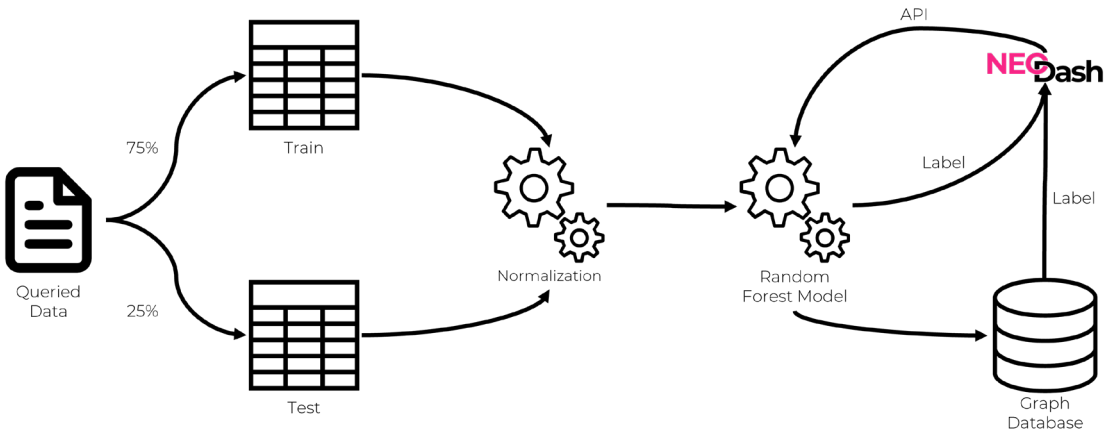
T.3 Confusion matrix of Random Forest model

orders of magnitude of some variables. In order to train the model, the scikit-learn package (Pedregosa et al., 2011) is used. To assess the performance of the model a confusion matrix is created as shown in Table 3. This matrix indicates a precision of 92%, recall of 92%, and overall accuracy of 89%. For the purposes of this use case, this performance is deemed acceptable and therefore the model will be used in the remainder of this project. As Figure 30 shows, the prediction model is used in two different ways. Firstly, the dashboard can access the model directly via a custom API. Here,

```
select ?building ?propertyName ?value ?energyLabel ?builtYear ?area ?purposeOfUse where {
    {
        select (sum(if(?purposeOfUse = 'woonfunctie', 1, 0)) as ?dwellings) (count(?building) as ?totalBuildings) ?neighborhood {
        ?building bag2:gebruiksdoel ?purposeOfUse.
        ?neighborhood bot:containsZone ?building
        } GROUP BY ?neighborhood
    }
    bind(?dwellings/?totalBuildings as ?percentage)
    FILTER (?percentage > 0.95)
    ?neighborhood bot:containsZone ?building.
    FILTER NOT EXISTS {?building a neo:Neighborhood}
    ?building bop:hasProperty ?energyLabelProperty.
    ?building bag2:bouwjaar ?builtYear.
    ?building bag2:oppervlakte ?area.
    ?building bag2:gebruiksdoel ?purposeOfUse.
    ?energyLabelProperty bop:hasValue ?energyLabel.
    OPTIONAL {
        ?neighborhood bag2:postcode ?postcode
    }
    OPTIONAL {
        ?neighborhood bot:containsZone ?pc.
        ?pc a neo:Neighborhood.
        ?pc bag2:postcode ?postcode.
    }
    ?neighborhood bop:hasProperty ?property.
    FILTER(?propertyName in ('incomePerRecipient', 'electricityUse', 'gasUse', 'WOZValue', 'powerGeneration'))
    ?property skos:prefLabel ?propertyName.
    ?property bop:hasExecution ?execution.
    ?execution bop:hasResult ?result.
    ?result seas:hasTemporalContext ?context.
    ?context time:hasBeginning/time:inXSDDateTimeStamp ?begin.
    FILTER (?begin = '2020-01-01T00:00:00'^^xsd:dateTimeStamp)
    ?result bop:hasValue ?value.
}
```

F.29 SPARQL query to collect data for predictive analysis

the API call includes the relevant data on an individual building and the return message indicates whether that building is more likely to have an energy label above or below C according the to model. An implementation of this is shown in Figure 31. This image shows how for each dwelling in the building the energy label is checked, which is shown if it is available. Otherwise, the API call is made and the results are shown. Considering that the known energy labels in this building are D, the result of 'Worse than C' seems reasonable. While this method works for a relatively small amount of dwellings, processing times become intractable if all 51099 dwellings need to be estimated in real-time every time an analysis is done. Therefore, the results of the RF model are also stored in the graph database directly, so they can be queried for the energy poverty analysis directly. The ontological structure for storing the energy labels is shown in Figure 34. Here it can be seen that the same structure for defining properties on a neighborhood level is used. This structure allows the user to differentiate between actual and predicted energy labels by querying the appropriate procedure (in Figure 34 a predicted energy label is shown). Section 4.1 has shown the LILEK analysis, however, this analysis was conducted using only the available, known energy labels. Using



F.30 Process for incorporating Random Forest Model into dashboard

Selected Building

Electricity Use Areas | Gas Use Areas | Neigbhorhood | Postal Code 4 | Postal Code 5 | Postal Code 6

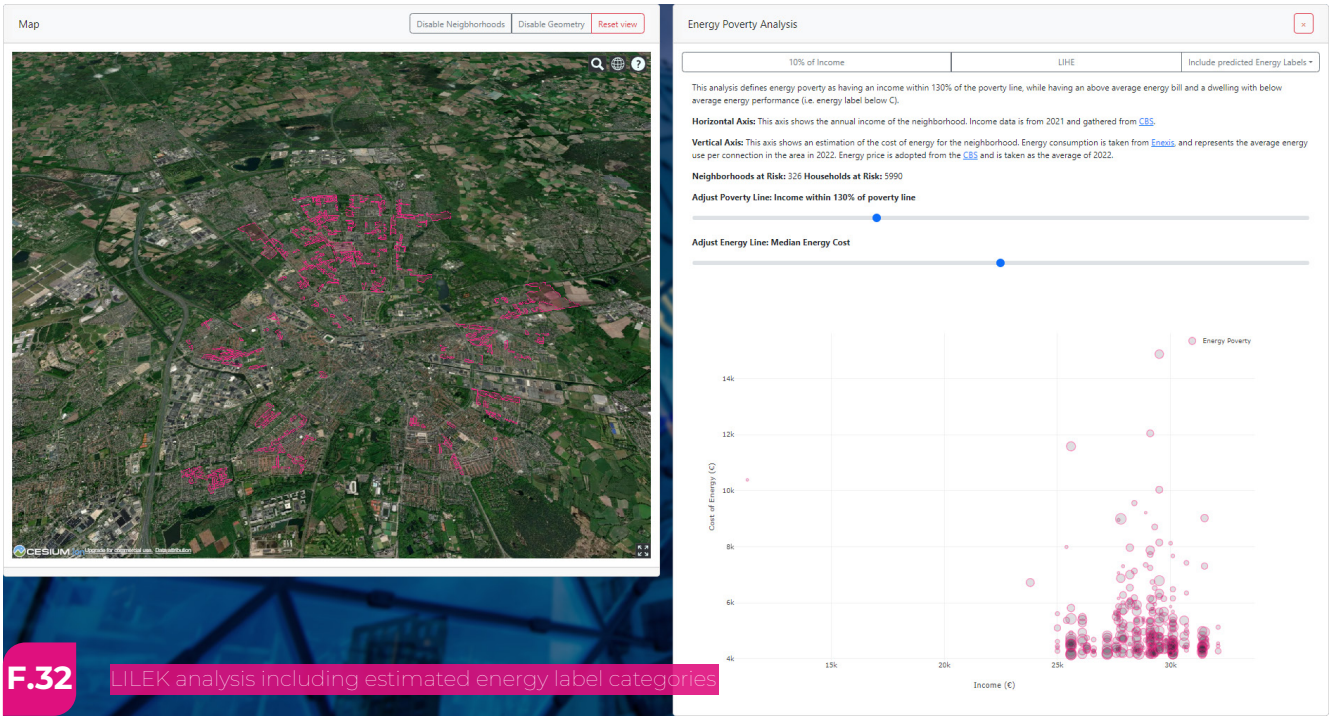
Energy Data

#	Postal-code	House Number	Area (m2)	Build Year	Purpose of Use	Energy Label
			75	1963	Woonfunctie	D
			75	1963	Woonfunctie	D
			75	1963	Woonfunctie	D
			75	1963	Woonfunctie	Worse than C
			75	1963	Woonfunctie	Worse than C
			75	1963	Woonfunctie	D

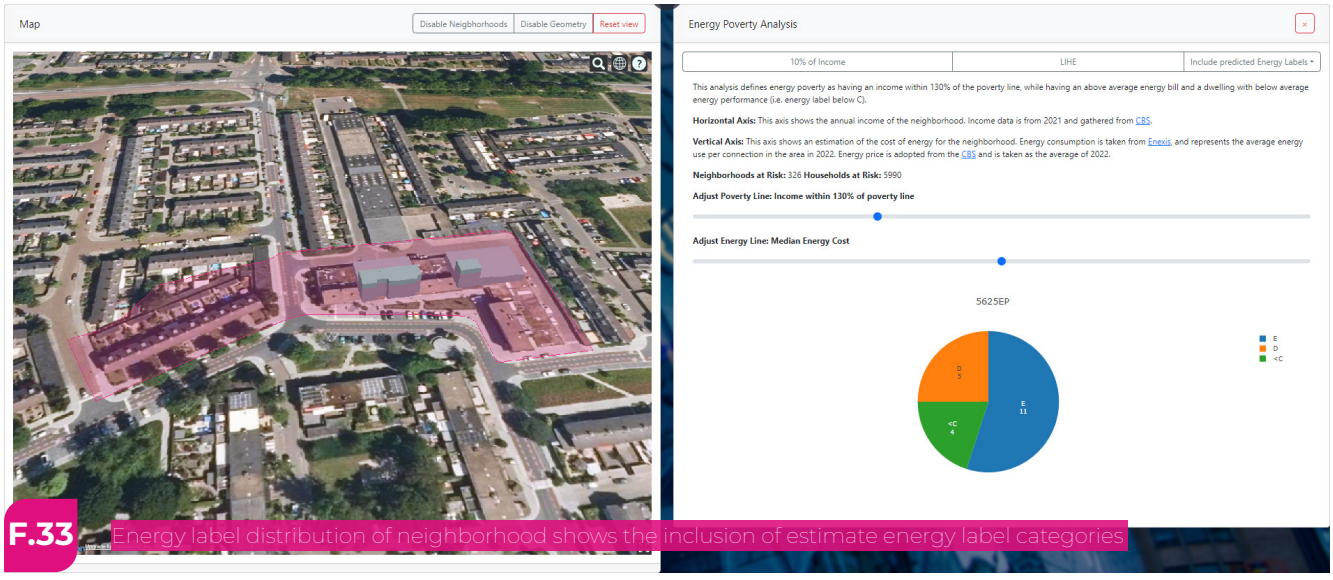
F.31 Results of API call made to Random Forest model

the estimated energy label categories, this analysis can be performed differently, yielding the results shown in Figure 32. While these results do not seem different from the results shown in Figure 20, Figure 33 shows that the newly estimated energy label categories are indeed included in the analysis. Further analysis shows that roughly 335 (5%) additional households are considered to be at risk according to the analysis including estimated energy label categories. This final use case has shown how the proposed data structure can be leveraged to gather data to train predictive models, while also retrieving data to make predictions. Moreover, these models can be integrated into the created dashboard which allows for more comprehensive analyses. While this use case serves as an example of classical machine learning, it might serve as a first step towards more robust artificial intelligence according to the definition provided by Marcus (2020). 'Business as usual has focused primarily on steadily improving tools for function approximation and composition within the deep learning toolbox, and on gathering larger training sets [...] one can imagine improving systems by gathering larger data sets, augmenting those data sets in various ways, and incorporating various kinds of improvements in the underlying architecture.' (Marcus, 2020). The author argues for an approach more similar to the cognitive cycle where humans take in perceptual information from the outside, build internal cognitive models based on their perception of that information, and then make decisions with respect to those cognitive models. Here, the proposed ontological structure might serve as a 'cognitive model' for decision-making in the urban setting, which would allow for more robust, contextual urban modeling by artificial intelligence in the future. Or, to build systems that an 'routinely acquire, represent, and manipulate abstract knowledge, using that knowledge in the service of building, updating, and reasoning over complex, internal models of the external world' (Marcus, 2020).

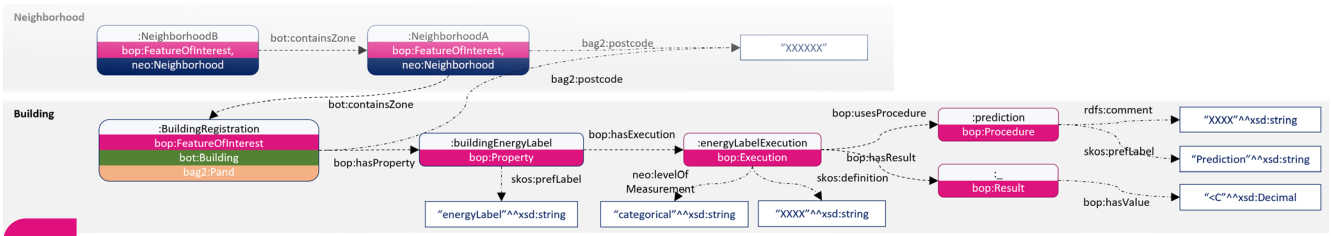
4. Use Cases - Predictive Analytics



F.32 LILEK analysis including estimated energy label categories



F.33 Energy label distribution of neighborhood shows the inclusion of estimate energy label categories



F.34 Ontological structure for storing energy labels

5. DISCUSSION & CONCLUSION

The goal of this project was to make urban (energy use) data more meaningful. The seaming lack of meaning in the current situation is suggested to be threefold First of all, a general lack of building information and other data limits the possibilities for cross-domain monitoring, simulation, and interventions (Curry et al., 2013). Secondly, current methods for urban energy modeling are mostly either top-down or bottom-up models, which limit the possibilities for analysis on multiple scale levels (Abbasabadi and Mehdi Ashayeri, 2019). Therefore, semantic web technologies and Linked Data are suggested as possible solutions to create more meaning in the data. However, this leads to the third and last impediment to more meaningful urban data, which is the lack of available ontological structures that are deemed suitable for the purposes of this project. In order to solve these challenges a new ontological structure has been proposed, Neighborhood Energy Ontology (NEO). This allows for the description of urban data properties on multiple spatial and temporal levels, where properties can be transitive between urban areas ('neighborhoods') that contain each other. Built upon this data structure is the semantic digital twin NeoDash. This digital twin allows the user to explore the available data in a more intuitive way. Moreover, several use cases have shown how the digital twin can be implemented to solve societal issues. The newly proposed ontological structure (Neighborhood Energy Ontology (NEO)) holds some advantages over previously mentioned ontologies. Ontologies like those mentioned by Chun et al. (2020); Reinisch et al. (2011); Daniele (2020), are mostly concerned with energy use on a single scale (microgrids, houses, and appliances). While this is suitable for those use cases, this methodology does not solve the issues mentioned by (Abbasabadi and Mehdi Ashayeri, 2019) of lack of analysis on multiple scale levels. Similarly, existing ontologies like SAREF4CITY and EM-KPI, are mostly concerned with larger-scale urban data. NEO aims to bridge the gap between these two approaches and allows for large-scale urban data (section 4.1 to smaller-scale building level (section 4.2 data. Moreover, as NEO is built upon the structure of both BOT and BOP, even smaller spatial levels could be achieved. This shows how this challenge of top-down and bottom-up integration is also addressed by NEO. As data is integrated on all scale levels, it can be aggregated from the top-down or the bottom-up, depending on the desired analysis. Different implementations of this were given in section 4.2. Furthermore, as the developed semantic digital twin is based upon NEO, it is suggested to hold an advantage over more traditional GIS platforms (based on, for example, cityGML). The conducted use cases show how the integrated data can be used for multiple types of analysis and models, where the user is no longer limited by a lack of integration. This allows to user to ask new, different, and more complex

questions that were harder or infeasible to ask in a more traditional urban model. In order to explore the possibilities of semantic digital twins further, the possibilities of ontology alignment should be further explored. As has been mentioned, existing ontologies have great potential to describe specific energyrelated topics, which could be beneficial for larger-scale urban data analysis. Therefore, alignment between NEO and ontologies like SAREF4CITY and EMKPI could prove beneficial. Moreover, while this project has focussed on the feasibility of developing a semantic digital twin, the implications of such a tool should be further investigated. Potential risks, such as privacy should be further explored. Section 4.2 has shown how more accurate energy use data can be used, however, it did not discuss the privacy implications of making such data available. Moreover, this might well depend on the end-user of such a tool, which has not been made explicit in this project. Depending on the end-user and their specific incentives, the societal, economic, and environmental impact of such a semantic digital twin should be further investigated. It is suggested that, while the data structure will not change, different digital twins can be built upon that data structure in order to serve different end-users with specific end goals. This assumption should be explored further in future work. Concluding, this project has aimed to add to the current state of urban energy modeling through the addition of semantic web technologies and Linked Data. The newly proposed ontology adds to the existing body of ontological structures by allowing for urban data description on multiple spatial and temporal levels. Moreover, the created semantic digital twin leverages this data structure to allow for more intuitive data exploration through visual means. The benefit of such a digital twin is shown through the use cases described in section 4. These use cases show how Linked Data in combination with digital twins can, indeed, provide additional and more meaningful insight into urban data.

REFERENCES

Abbasabadi, N., Ashayeri, M., Azari, R., Stephens, B., and Heidarinejad, M. (2019). An integrated data-driven framework for urban energy use modeling (UEUM). Applied Energy, 253.

Abbasabadi, N. and Mehdi Ashayeri, J. K. (2019). Urban energy use modeling methods and tools: A review and an outlook.

Ali, U., Shamsi, M. H., Alshehri, F., Mangina, E., and O'Donnell, J. (2019). Application of intelligent algorithms for residential building energy performance rating prediction. In Building Simulation Conference Proceedings, volume 5, pages 3177–3184. International Building Performance Simulation Association.

Ali, U., Shamsi, M. H., Bohacek, M., Purcell, K., Hoare, C., Mangina, E., and O'Donnell, J. (2020). A data-driven approach for multi-scale GIS-based building energy modeling for analysis, planning and support decision making. Applied Energy, 279.

Ali, U., Shamsi, M. H., Hoare, C., Mangina, E., and O'Donnell, J. (2021). Review of urban building energy modeling (UBEM) approaches, methods and tools using qualitative and quantitative analysis.

Amber, K. P., Aslam, M. W., and Hussain, S. K. (2015). Electricity consumption forecasting models for administration buildings of the UK higher education sector. Energy and Buildings, 90:127–136.

CBS (2019). Kerncijfers per postcode.

Centraal Bureau voor de Statistiek (2022). 1 op de 4 mensen met armoederisico is een kind.

Cesium (2022). Cesium - The Platform for 3D Geospatial.

Cesium (2023). Cesium Ion.

Chun, S., Jung, J., Jin, X., Seo, S., and Lee, K. H. (2020). Designing an integrated knowledge graph for smart energy services. Journal of Supercomputing, 76(10):8058–8085.

Corry, E., Pauwels, P., Hu, S., Keane, M., and O'Donnell, J. (2015). A performance assessment ontology for the environmental and energy management of buildings. Automation in Construction, 57:249–259.

Curry, E., O'Donnell, J., Corry, E., Hasan, S., Keane, M., and O'Riain, S. (2013). Linking building data in the cloud: Integrating cross-domain building data using linked data. Advanced Engineering Informatics, 27(2):206–219.

Daniele, L. (2020). SAREF4ENER: an extension of SAREF for the energy domain created in collaboration with Energy@Home and EEBus associations.

Daniele, L., den Hartog, F., and Roes, J. (2015). Created in Close Interaction with the Industry: The Smart Appliances

REFerence (SAREF) Ontology. In Lecture Notes in Business Information Processing, volume 225, pages 100–112. Springer Verlag.

De Nicola, A. and Villani, M. L. (2021). Smart city ontologies and their applications: A systematic literature review.

Degha, H. E., Laallam, F. Z., and Said, B. (2019). Intelligent context-awareness system for energy efficiency in smart building based on ontology. Sustainable Computing: Informatics and Systems, 21:212–233.

Department for Business, E. . I. S. (2020). Fuel Poverty Methodology Handbook (Low Income High Costs). Technical report.

Department for Energy Security & Net Zero (2023). Fuel Poverty Methodology Handbook (Low Income Low Energy Efficiency). Technical report.

Donkers, A., Yang, D., De Vries, B., and Baken, N. (2021). Building Performance Ontology. Revision: 1.6

Enexis Netbeheer (2023). Open Data.

Hausenblas, M. and Kim, J. (2012). 5-Star Open Data.

HomeWizard BV (2023). Wi-Fi P1 Meter.

Kadaster (2021). Basisregistratie Adressen en Gebouwen.

Kim, T. Y. and Cho, S. B. (2019). Predicting residential energy consumption using CNN-LSTM neural networks. Energy, 182:72–81.

Kolter, J. Z. and Ferreira, J. (2011). A Large-Scale Study on Predicting and Contextualizing Building Energy Usage. Technical report.

Li, W., Zhou, Y., Cetin, K., Eom, J., Wang, Y., Chen, G., and Zhang, X. (2017). Modeling urban building energy use: A review of modeling approaches and procedures.

Li, Y., Garcıa-Castro, R., Mihindukulasooriya, N., O'Donnell, J., and Vega-Sanchez, S. (2019). Enhancing energy management at district and building levels via an EM-KPI ontology. Automation in Construction, 99:152–167.

Marcus, G. (2020). The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence

Moore, R. (2012). Definitions of fuel poverty: Implications for policy. Energy Policy, 49:19–26.

Mulder, P., Batenburg, A., and Dalla Longa, F. (2023). Energiearmoede in Nederland 2022. Technical report.

Ontotext (2023). GraphDB by Ontotext.

Pauwels, P., Zhang, S., and Lee, Y. C. (2017). Semantic web technologies in AEC industry: A literature overview.

Pedregosa, F., Michel, V., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Vanderplas, J., Cournapeau, D., Pedregosa, F., Varoquaux, G., Gramfort, A., Thirion, B., Grisel, O., Dubourg, V., Passos, A., Brucher, M., Perrot, M., Duchesnay, ˜A., and Duchesnay, F. (2011). Scikit-learn: Machine Learning in Python. Technical report.

Peters, R., Dukai, B., Vitalis, S., Van Liempt, J., and Stoter, J. (2021). Automated 3D reconstruction of LoD2 and LoD1 models for all 10 million buildings of the Netherlands. Technical report, Delft

University of Technology.

Rasmussen, M. H., Lefrançois, M., Schneider, G. F., Pauwels, P., and Janowicz, K. (2020). BOT: the Building Topology Ontology of the W3C Linked Building Data Group. Technical report.

Poveda-Villalon, M., Garcia-Castro, R., and Espinoza-Arias, P. (2020). SAREF extension for Smart City.

Reinisch, C., Kofler, M. J., Iglesias, F., and Kastner, W. (2011). Thinkhome energy efficiency in future smart homes. Eurasip Journal on Embedded Systems, 2011.

Rijksdienst voor Ondernemend Nederland (RVO) (2022). Openbare data energielabels.

United Nations Department of Economics and Social Affairs (2019). World Population Prospects 2019.

VanDerHorn, E. and Mahadevan, S. (2021). Digital Twin: Generalization, characterization and implementation. Decision Support Systems, 145.

vis.js (2023). Vis.js community edition.

Wang, W., Lin, Q., Chen, J., Li, X., Sun, Y., and Xu, X. (2021). Urban building energy prediction at neighborhood scale. Energy and Buildings, 251.

Zhao, H. X. and Magoules, F. (2012). A review on the prediction of building energy consumption.

Images

Cover & page 41
Marien, R. (2021) https://unsplash.com/photos/xRcd-dI_a7tw

Page 2-3
Ziajowska, A. (2022) https://unsplash.com/photos/tWRKNpWPVaw

Page 4-5
Quinten, J. (2021) https://unsplash.com/photos/YsVh-cvoJp0k

Page 6-7
Fernándex, F. (2019) https://unsplash.com/photos/ibsX-mc68psk

Page 8
Ijsendoorn, P., PhotosPublic (n.d.)

Page 12
ruddy.media (2019) https://unsplash.com/photos/sN8L-GWF7A0I

Page 16
Cotimani, A. (2020) https://unsplash.com/photos/pGLeXbxeBCc

Page 22
rawPixel, (n.d.) https://www.rawpixel.com/image/3370024/free-photo-image-glowing-amusement-park-architecture

Page 36
Kohler, T. (2022) https://unsplash.com/photos/Xjqxrxn-Hjbw

Page 38
Ram, O. (2020) https://unsplash.com/photos/RsTgMxS-FppI

Page 42
Kunnen, Bram (2017) https://unsplash.com/photos/PFeFOCB6S18

Page 60
Heijmerikx, R. (2020) https://unsplash.com/photos/nSCx6_pQdIU

Page 103
BNW (2020) https://unsplash.com/photos/8dmuy_M7t9g

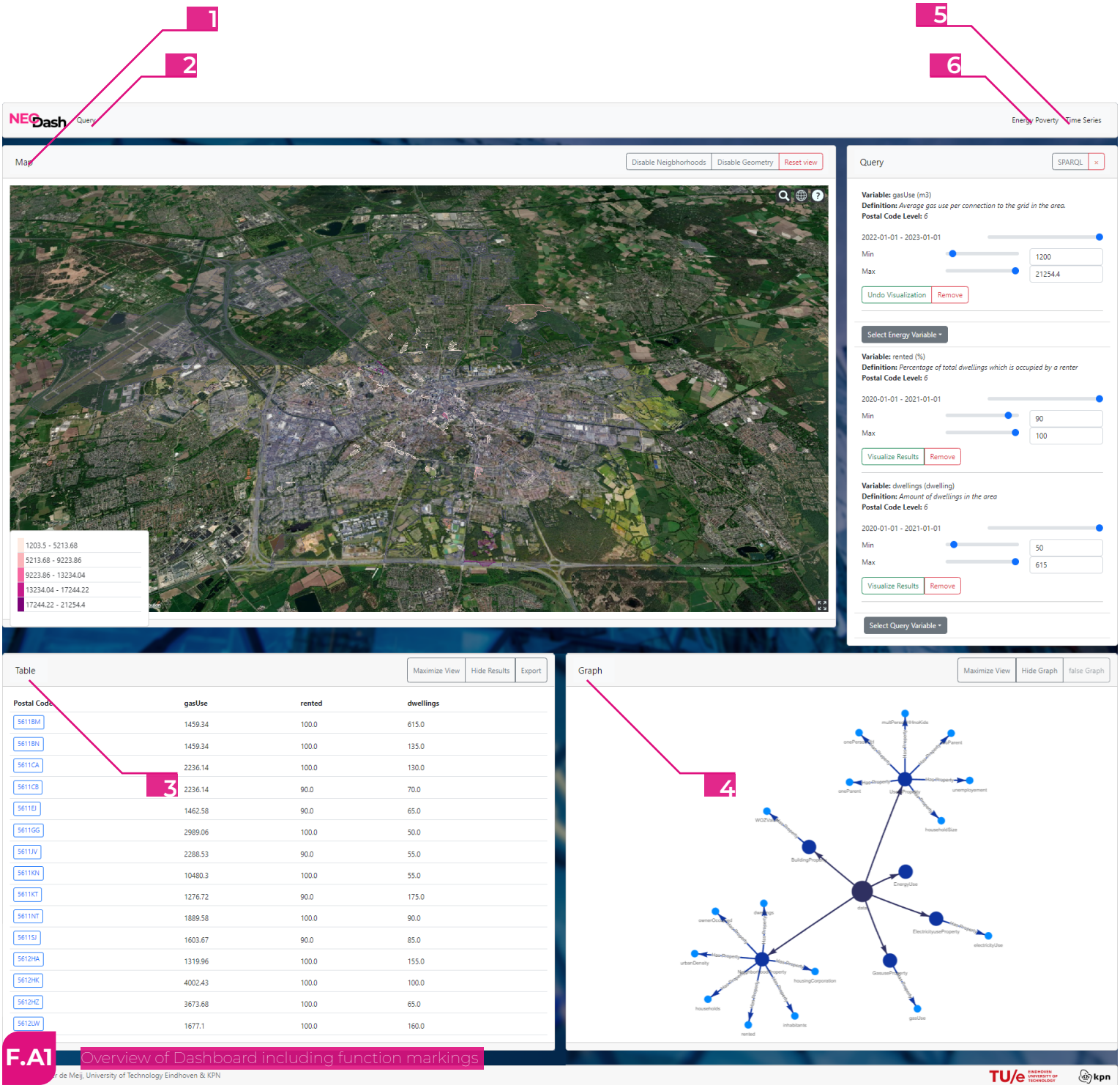
Design based on: https://visme.co/blog/business-report-templates/

Appendix A Dashboard

This appendix describes all functionalities of the dashboard in more detail. Below every aspect of the dashboard will be discussed, including their assigned names, uses and which functions they include. These functions are part of the code which builds up the dashboard, which are further explained in Appendix B (page 60). Firstly, the major parts of the dashboard will be explained as in Section 3 of the main report. After which, each part of the dashboard will be explained in more detail. While the main report gives an overview of the most important functions, this Appendix will explain all functionalities build in to the dashboard and why they might be relevant for the overall function.

#	Name	Use	Functions	Code
1	Map	As explained in Section 3.1, the main functionality of the dashboard is the map. The map visualizes the city under investigation and the results of the queries and analysis when relevant. All functions will be detailed in A.1 below.	loadCesium() toggleGeometry() toggleNeighborhoods() toggleAll(false)	B.1 B.2 B.3 B.4
2	Query Button	Toggles the Query Box, which is elaborated in Section 3.2 and in A.2 below.	toggleAll('sideColumn')	B.4
3	Table	Shows the relevant results of the query and has interactive functions with the map (Section 3.3, A.3).	updateTable(postalCode) orderTable(variable, ordering)	B.5 B.6
4	Graph	Shows the relationship of the queried variable or neighborhood in graph format (Section 3.4, A.4).	getMetaData() getInstanceData()	B.7 B.8
5	Time Series Button	Toggles the Time Series Analysis Box (Section 4.2, A.5).	toggleAll('timeSeriesBox')	B.4

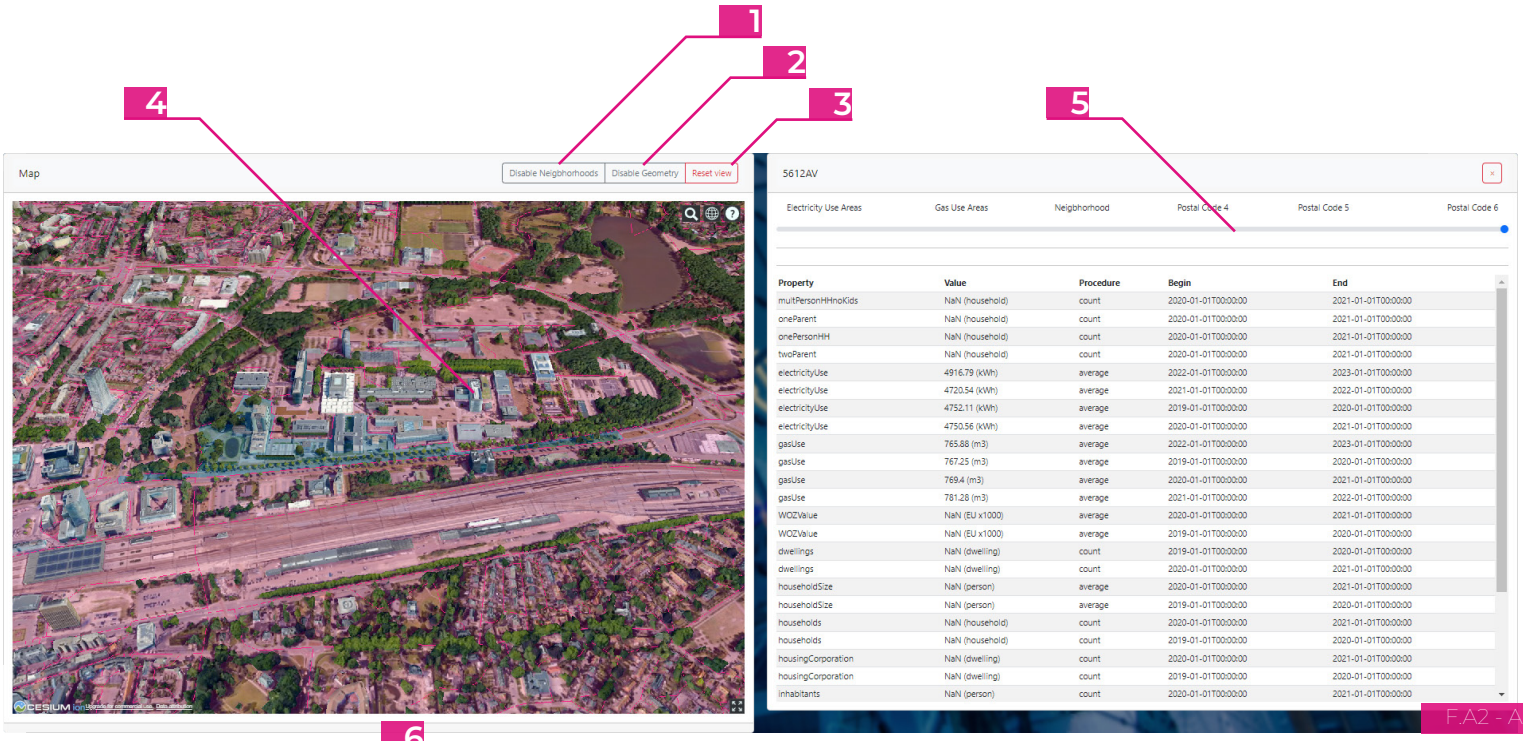
#	Name	Use	Functions	Code
6	Energy Poverty Button	Toggles the Energy Poverty Analysis Box (Section 4.1, A.6).	toggleAll('energyPoverty-Box')	B.4

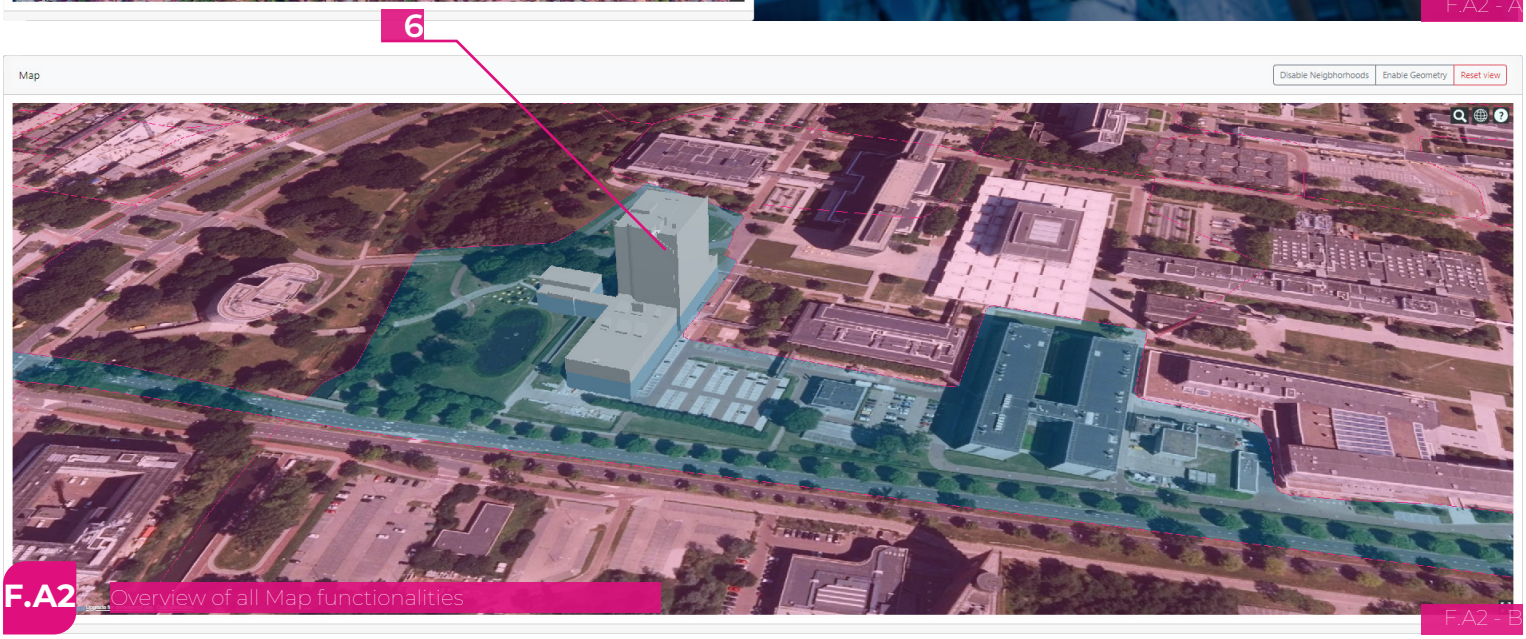


F.A1 Overview of Dashboard including function markings

A.1 Map

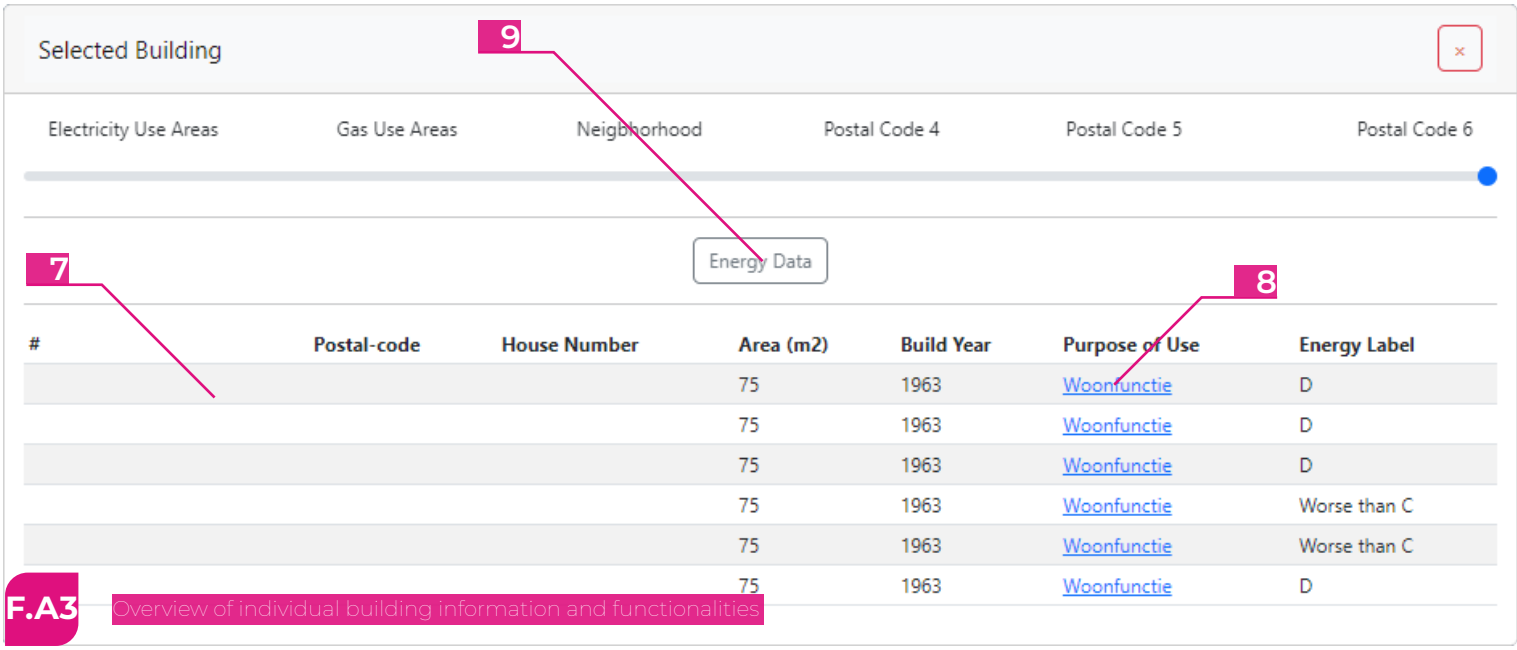
#	Name	Use	Functions	Code
1.1	Disable Neighborhoods Button	The user is able disable (and enable) the visualization of the neighborhoods. As the neighborhoods present a strong visual component of the map, the user might choose to not visualize them. This also disables the interactivity on building level.	toggleNeighborhoods()	B.3
1.2	Disable Geometry Button	Similar to function 1.1, this button disables (or enables) the LiDAR geometry, visualizing the BAG volumes seperatly (Figure A2-B). This might assist the user in selecting the appropriate building.	toggleGeometry()	B.2
1.3	Reset View Button	Some queries and interactions might lead to upredictable results or an error in the dashboard. Therefore, the user is able to reset the view (and current queries) as to start their analysis over.	resetView()	
1.4	Cesium Map	The main functionalities have been explained in Section 3.1. The user is able to perform traditional 3D manipulations like panning and zooming. Moreover, the standard Cesium geo-encoder (geospatial search bar) is functional. Dashboards specific functions are interaction with the neighborhoods and corresponding buildings. Interacting with the neighborhood results in a view as shown in Figure A2-A where the neighborhood is highlighted and the corresponding neighborhood data is shown. Moreover, this interaction enables interaction on a building level.	loadCesium() loadInfo(postalCode) togglePC(postalCode, building)	B.1 B.12 B.9





F.A2 Overview of all Map functionalities

#	Name	Use	Functions	Code
1.5	Spatial Level Slider	As has been explained in Section 3.1, the user is able to visualize neighborhood specific data on different spatial levels. This slider shows the available spatial levels and the user can select any of them by moving the slider, which updates the table below to only show data which is measured with that spatial resolution.	showArea(postalCode)	B.5
1.7	Building URI	When a building is selected, the user is able to link directly to the Cadastre data using the buildings URI. These numbers are hidden in Figure A3 for privacy reasons.		
1.8	Building Function	When a building is selected, the user is able to directly link to the Cadastre definition of the purpose of that building.		
1.9	Energy Data Button	The user is able to query whether time-series energy use is available for the selected building. If such data is available, the visualization shown in Figure 24 is created for the selected building.	loadBuildingEnergyUse()	B.16

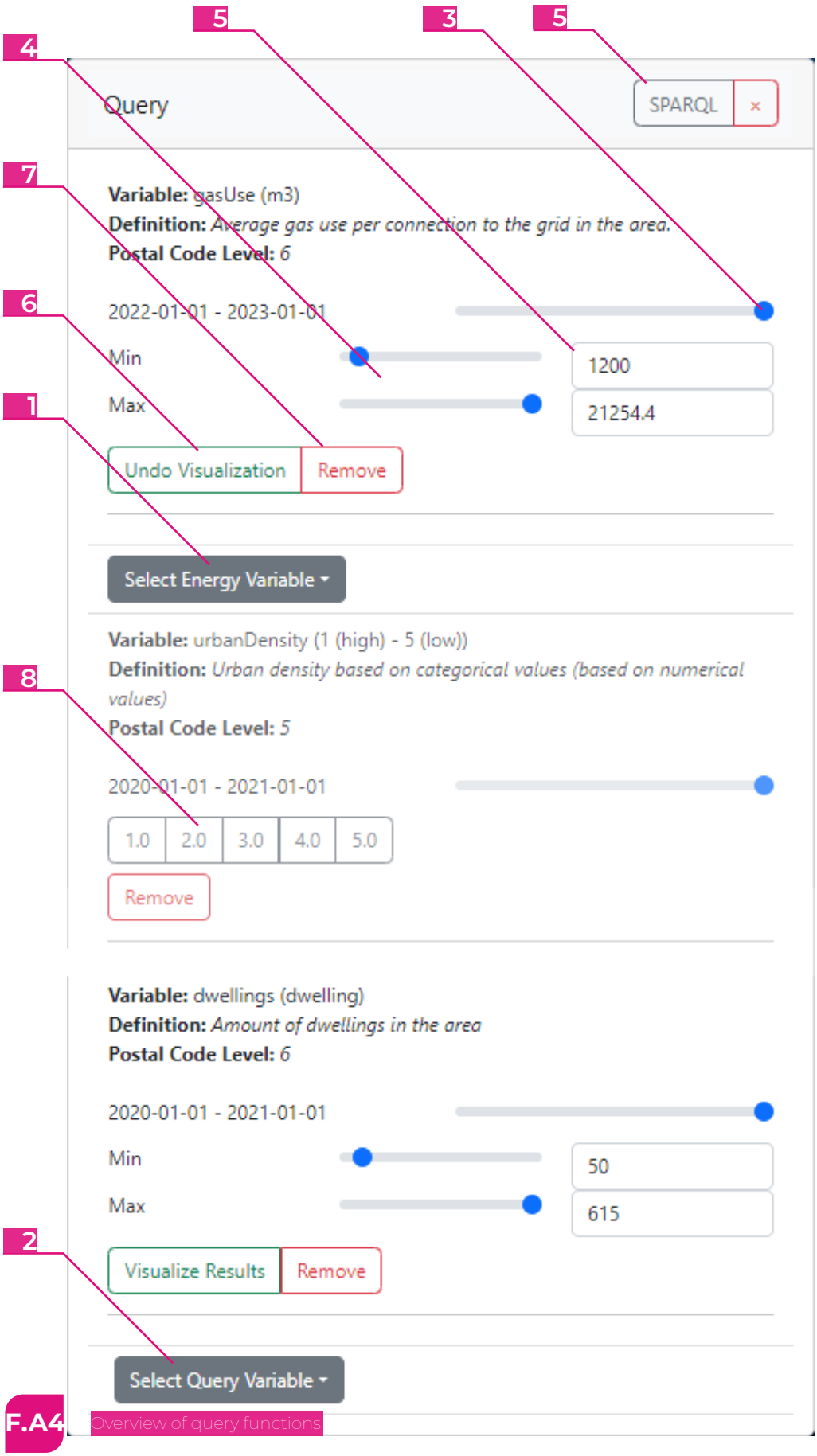


F.A3 Overview of individual building information and functionalities

A.2 Query

#	Name	Use	Functions	Code
2.1	Select Energy Variable Dropdown	The user is able to select any variable to query energy use in the dashboard. However, the tool suggest appropriate variables based on the neo:EnergyUse class defined in Figure 3. These variables are directly queried from the database (see Figure 10).	getQueryVariables(energy)	B.17
2.2	Select Variable Dropdown	Same function as described in 2.1, however, the tool gives no recommendations for energy use specific variables	getQueryVariables(energy)	B.17
2.3	Time Frame Slider	The slider lets the user select the appropriate time frame related to that specific variable. By default, the most recent time frame is selected. Moreover, the user is able to evaluate different variables on different time frames.	updateRatioSlider(variable, minMax, inputType, procedureName) updateDateSlider(variable)	B.18 B.19
2.4	Variable Range Sliders	The user is able to select the range of values they want to query. The range of these values is defined by the found values. Moreover, the opposing slider is automatically adjusted when the slider exceeds that opposing slider.	updateRatioSlider(variable, minMax, inputType, procedureName) updateRatioValue(variable, inputType)	B.18 B.20
2.5	Variable Range Input	Similar to function 2.4, the user is able to manually input the range of the queried variable.	updateRatioSlider(variable, minMax, inputType, procedureName) updateRatioValue(variable, inputType)	B.18 B.20
2.6	Visualization Button	For the appropriate variables, the user is able to visualize the range of results found by the set query. Here, the found values are 'binned' into five categories (from low to high) and an appropriate color is connected to each category. These cateogories are then visualized on the map (as shown in Figure 6).	visualizeScore(variable, showArea)	B.13

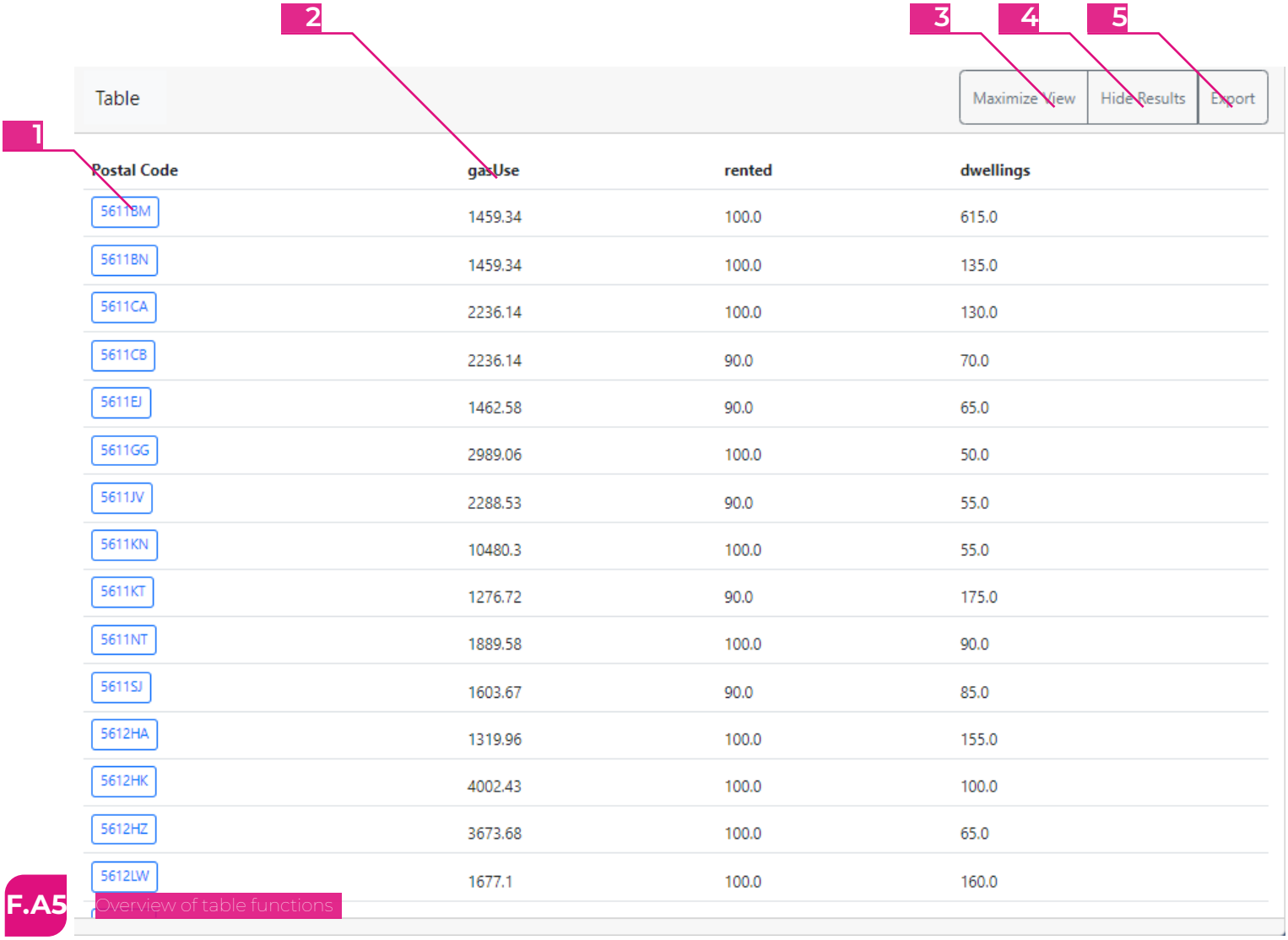
#	Name	Use	Functions	Code
2.7	Remove Variable Button	If the user no longer wants to include a variable in their query or analysis they can remove it via this button. The query is automatically updated and possible visualizations are removed.	removeVariable(variable)	B.21
2.8	Category selection buttons	Similar to function 2.4, the user is able to select one of the found categories describing this variable. Differing variable representations are chosen as described in Section 3.2,	onButtonSelect(variable, category, procedureName)	B.22



F.A4 Overview of query functions

A.3 Table

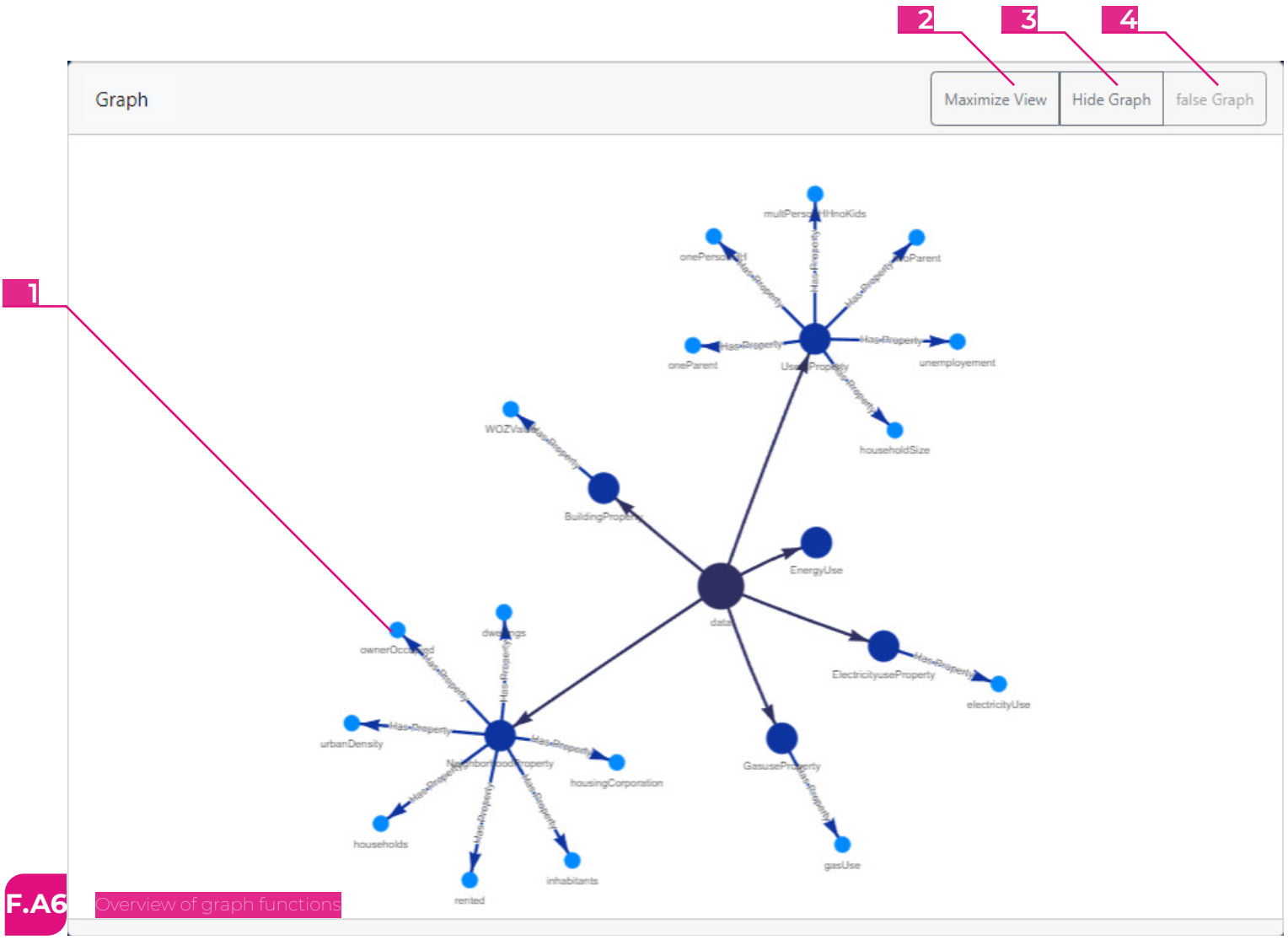
#	Name	Use	Functions	Code
3.1	Neighborhood Button	The user is able to select a specific neighborhood from the results. This neighborhood is then automatically visualized in the map.	showSinglePostalCode(postalcode)	B.23
3.2	Order Table	The user can order to table from high to low or low to high when the corresponding variable is clicked in the results table.	orderTable(variable, ordering)	B.6
3.3	Maximize View Button	As can be seen in Figure 6, the results table only fills halve of the width of the screen by default. However, the user can choose to display the table on the full width of the screen by pressing this button.	fullScreen()	B.24
3.4	Hide/Show Results Button	The default state of the results table is 'hidden', meaning that the results are not shown in order not to clutter the dashboard. When the user wants to see the results table, they can press this button to show it, after which they can use this button again to hide the results table again.	showResults()	B.25
3.5	Export Button	If the user wants to use the results of the query in another application or tool, they can export the results table in `.csv' format by pressing this button.	exportData()	B.26



F.A5 Overview of table functions

A.4 Graph

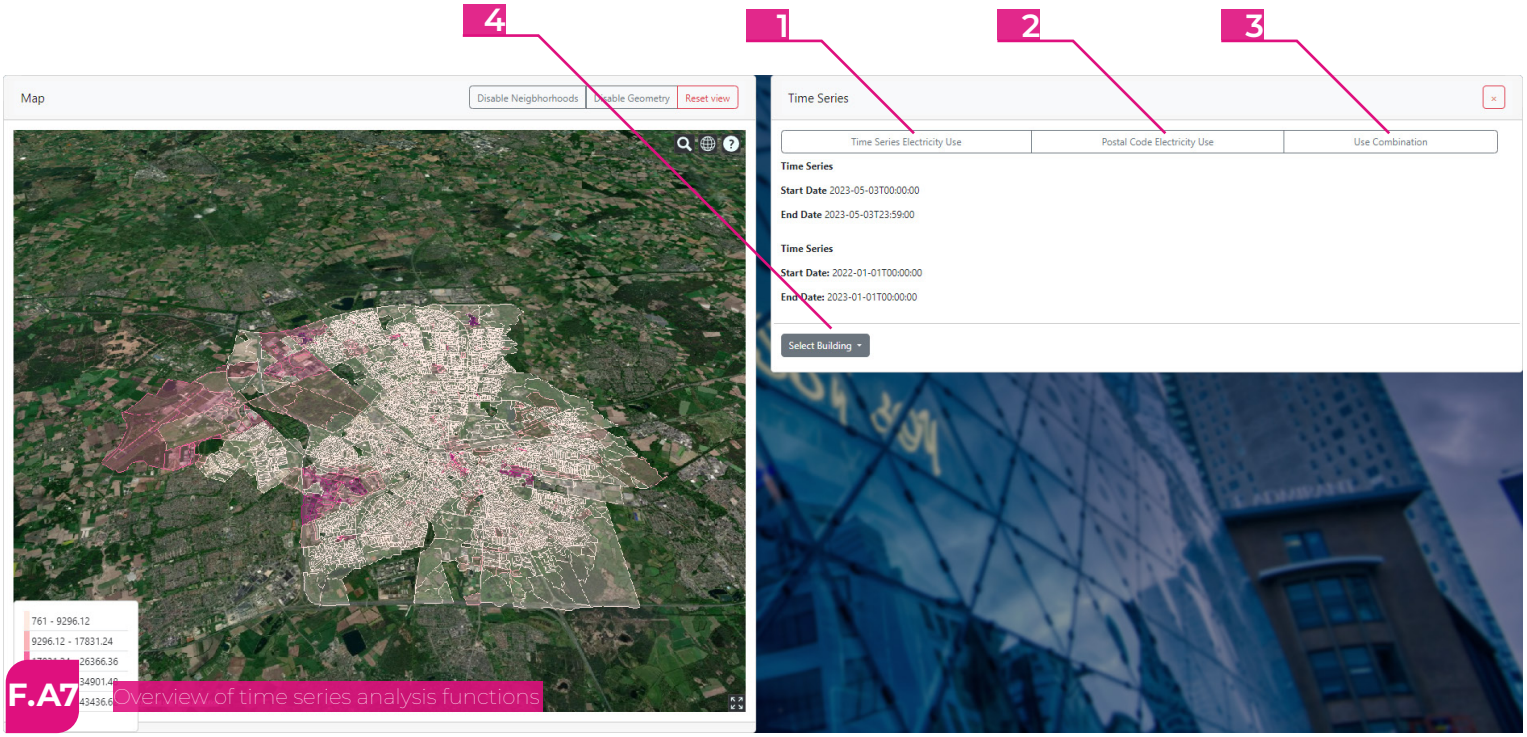
#	Name	Use	Functions	Code
4.1	Graph Interaction	By default, the 'metadata' of the available data is represented in the graph. However, in order to avoid unnecessary cluttering, not all information is shown. Therefore, the user is able to interacht with the graph to show more information, specific to the user's interest.	getMetaData()	B.7
4.2	Maximize View Button	Similar to function 3.3, the graph visualization only covers half of the width of the screen by default. If the users wants to visualize the graph on the full width, they can do so by pressing this button.	fullScreen()	B.24
4.3	Hide/show Graph Button	Similar to function 3.4, the user can hide (minimize) the graph by pressing this button. When the graph is already minimized, the user can show it by pressing this button as well.	showGraph(graphType)	B.27
4.4	Neighborhood Graph Button	When a specific neighborhood is selected (see functions 1.4 and/or 4.1), the specific data for that neighborhood is visualized in this graph. The user can manually switch between the metadata and neighborhood specific graphs using this buton	showGraph(graphType)	B.27



F.A6 Overview of graph functions

A.5 Time Series

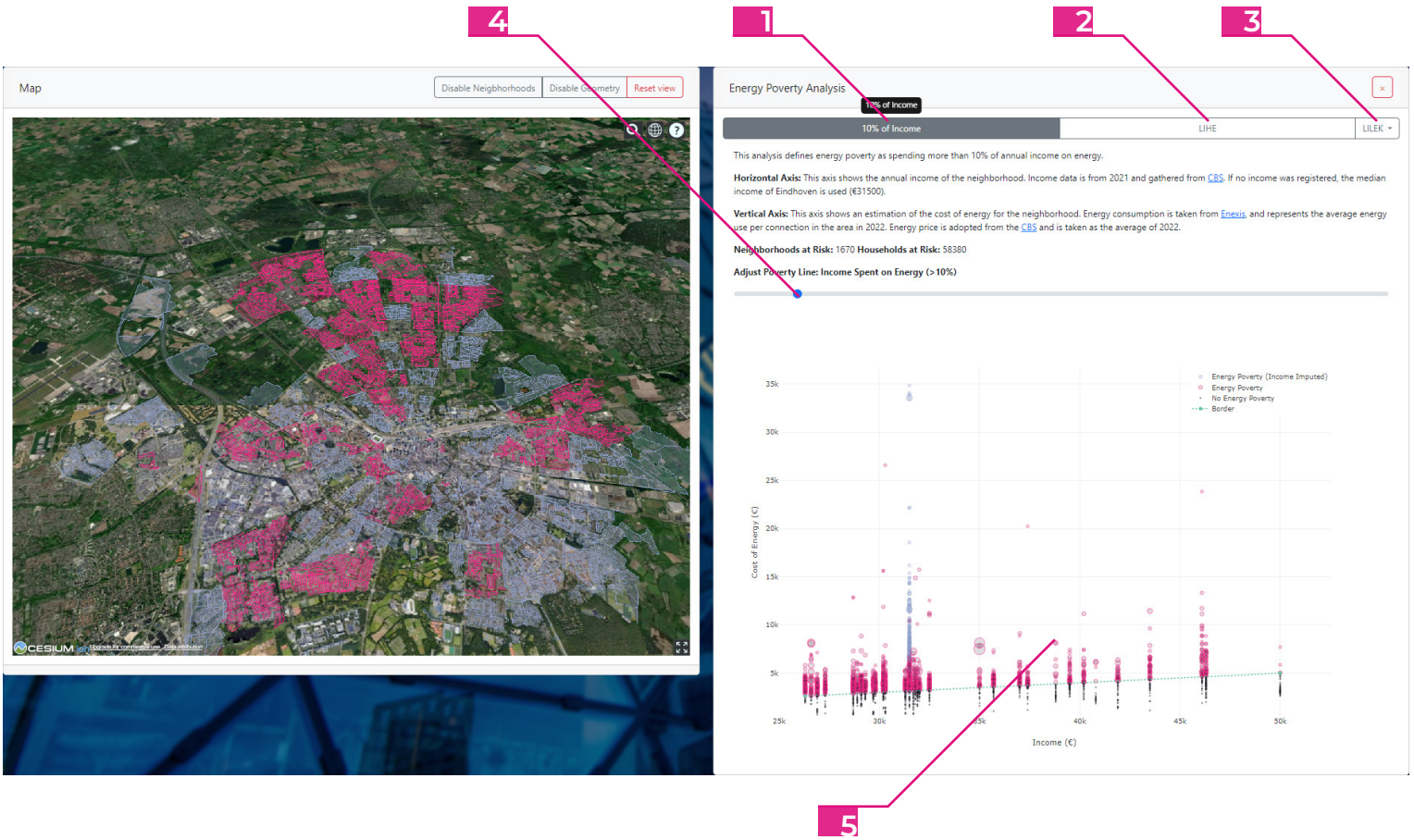
#	Name	Use	Functions	Code
5.1	Time Series Electricity Use Button	Within the time series analysis three main functions exists (functions 5.1, 5.2 and 5.3) The first of which is to evaluate which buildings have time series data attatched to them and use that as the source for the energy use visualization in the map (see Section 4.2, Figure 25).	getTimeSeries(level)	B.28
5.2	Postal Code Electricity Use	The second function within this analysis uses electricity use as it has been described previously, which is no different from the selection of electricity use in function 2.1.	getTimesSeries(level)	B.28
5.3	Combination Button	The third function combines the previous two functions and visualizes a combination of of the two different sources of electricity use. The advantages and disadvantages have been discussed in detail in Section 4.2.	getTimeSeries(level)	B.28
5.4	Select Building Dropdown	The final functionality within this analysis is to query a specific building which has either time series or real-time data associated with it in the graph database. This dropdown shows all those buildings. When a building is selected the visualization shown in Figure 28 is created.	getRTBuildings() queryDBrealTime(id, params)	B.29 B.33



F.A7 Overview of time series analysis functions

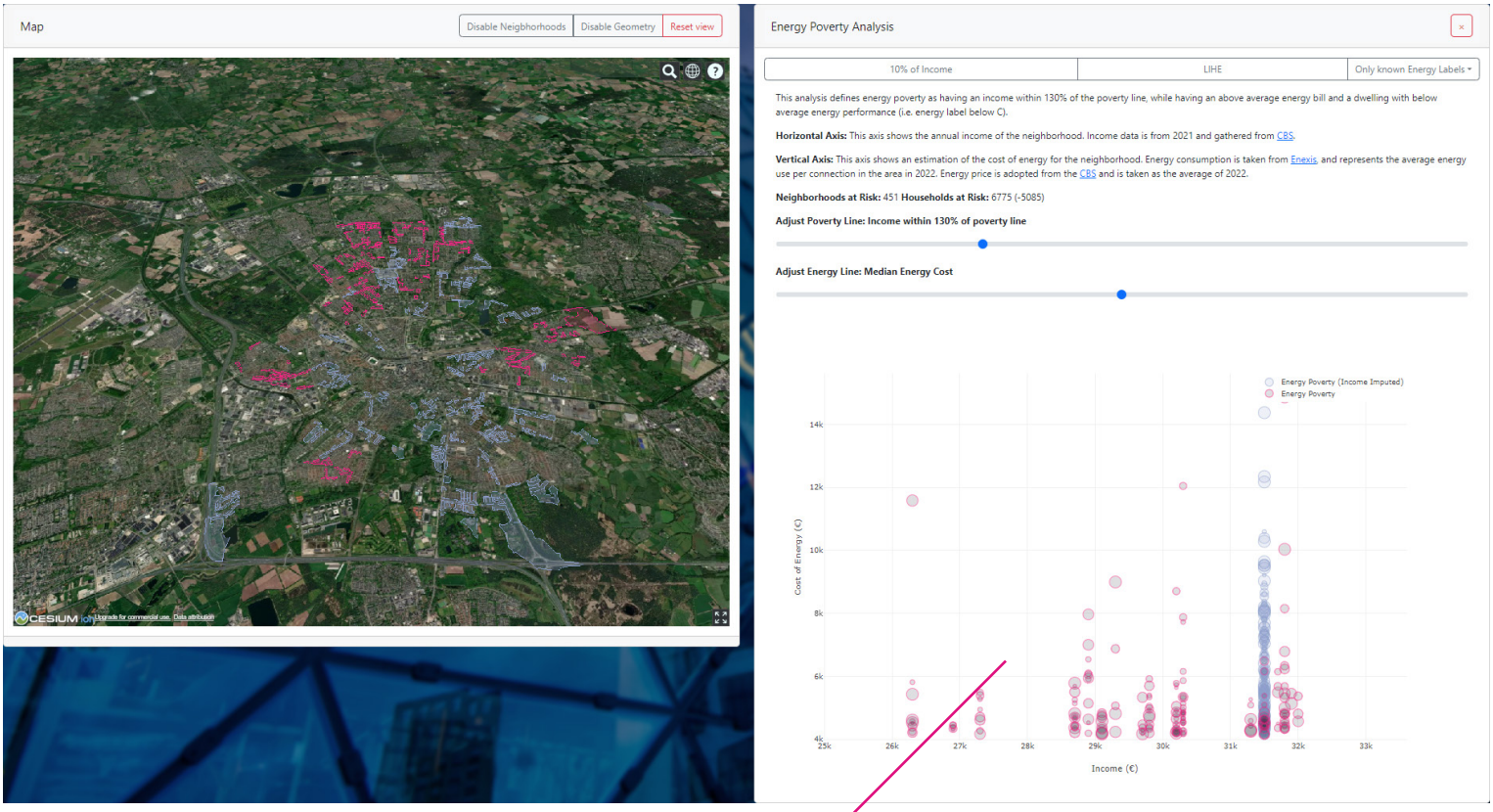
A.6 Energy Poverty

#	Name	Use	Functions	Code
6.1	10% of Income Button	As described in Section 4.1, this project considers three definitions of energy povert. The user is able to select to analyze the first of these definitions through this button. This will query the correct data and create a graph showing the estimated income and energy cost of all neighborhoods in the city. The neighborhoods which are considered energy poor according to this definition are visualized in the map (corresponding to the graph). Where income data was lacking, the median value of the city is imputed.	createEPGraph(toggle, epType, prediction)	B.30
6.2	LIHE Button	This button will show the second energy poverty defintion as defined in Section 4.1. The functionality is similar to function 6.1, however as the definition of energy poverty is different a slightly different graph is created (see Figure A8).	createEPGraph(toggle, epType, prediction)	B.30



F.A8 Overview of functionalities energy poverty analysis (10% of income and LIHE definitions)

#	Name	Use	Functions	Code
6.3	LILEK Dropdown	As the last definition of energy poverty is dependent on building level data (energy labels) this graph is created slightly differently (Figure A9). In this graph only neighborhoods where dwellings are considered at risk of energy poverty are shown, where the radius of the dot represent the relative amount of such dwellings. As is discussed in Section 4.3, the user can select whether to use only known energy labels, or also predicted energy labels for this analysis.	createEPGraph(toggle, epType, prediction)	B.30
6.4	Adjust Income Slider	As some neighborhoods are close to the border of energy poverty, the user is able to slightly adjust the definition using this slider. This will add or remove neighborhoods from the set of labelled neighborhoods, both in the graph as well as the map. This allows the user to investigate the sensitivity of the defintion.	updateAPLText('energy') createEPGraph(toggle, epType, prediction)	B.31
6.5	Graph Interaction	When a user interacts with one of the dots representing a neighborhood, the map will zoom in to that neighborhood. Moreover, the user is able to use generic functions in the graph such as: panning, zooming, resetting the axis and toggling datasets.		
6.6	Adjust Energy Cost Slider	As the second defintion of energy poverty relies on two borders (see Section 4.1), this additional slider allows the user to adjust the full definition of energy poverty in this scenario similar to function 6.4.	updateAPLText('energy') createEPGraph(toggle, epType, prediction)	B.31



F.A9 Overview of functionalities energy poverty analysis (LILEK definition)

#	Name	Use	Functions	Code
6.7	LILEK Graph Interaction	While function 6.5 describes interaction with the graph for the first two defintions of energy poverty, this function is specific to the graph of the last definition (LILEK). While it includes the functionality described in 6.5, it also generates a pie chart visualizing the different energy labels occuring in that neighborhood (see Figure 33). If the user clicks on one of the labels in this pie chart, the corresponding buildings in the neighborhood which have such a label will be highlighted.		

Appendix B Code

This appendix will give a psuedo code overview of the most important pieces of code of the dashboard described in this report. For each element of code the name of the function will be given, while the input variables will be defined. Lastly, the code will be explained through psuedo-code and textual explanation where necessary.

Moreover, within the code there are three important dictionaries which will be used throughout this appendix:

1. postalCodeDict - This dictionary is formatted as follows:

```
{
        variable 1 : [postalCode 1, ..., postalCode N]
        ...
        variable M: [postalCode 1, ..., postalCode N]
}
```

   In this dictionary all queried variables and their found postal codes will be stored. As will be explained later, this allows for the final visualization of all overlapping postal codes in the map function.
2. nanDict - This dictionary is formatted the same as the postal code dictionary (point 1), however, this dictionary stores all postal codes which have return an unknown value for that specific variable.
3. valuesDict- This dictionary is formatted as follows:

```
{
    variable 1 :
        {
            postalCode 1 : value,
            ...,
            postalCode N: value
        },
    ...,
    variable M:
        {
            postalCode 1: value,
            ....,
            postalCode N: value
        }
}
```

   This dictionary describes each specific value for each found postal code for that variable.

B.1 loadCesium()

Dependencies

togglePC(postalCode, building) B.9, p. 69
predictEnergyLabel(buildingNumber, postalCode, buildYear, area) B.10, p. 70

Query

- C.1, p. 104

Code

```
1: if viewer is loaded then
2:    load neighborhoods geometry
3:    load BAG geometry
4:    load LiDAR geometry
5:    enable on click functionalities
6:    if neigbhorhood is clicked then
7:       togglePC(postalCode, building)
8:    else
9:       Query Cadastre for building information
10:      Create data table containing building information
11:      supplement data table with energy lable using predictEnergyLa-
         bel(buildingNumber, postalCode, buildYear, area)
12:      togglePC(postalCode, building)
13:   end if
14: end if
```

This function loads the cesium software and the required data which is fundamental to the map visualization. Each data element is loaded as a layer, while interactivity with those layers is enabled. When the users clicks on the map it is checked whether this click has occured on the neighbhorhood geometry, if so, the neighborhood is toggled. Otherwise (assuming a layer is clicked), it assumed that the click has occured on a building. Therefore, that buildings data is queried directly from the Cadastre and the energy label of that building is retrieved.

B.2 toggleGeometry()

Code

```
for cesium layers do
    if the layer is the LiDAR layer then
        if geometry is disabled then
            Show LiDAR scan
        else if geometry is enabled then
            Hide LiDAR scan
        end if
    end if
end for
if geometry is enabled then
    if neighborhood is selected then
        Show BAG buildings in neighborhood
    else
        Load all BAG buildings
    end if
else
    hide BAG buildings
end if
```

The goal of this function is to show the BAG geometry. Therefore, the LiDAR scan needs to be disabled and the BAG geometry enabled (or reversed when the function is toggled again). Therefore, the function finds the LiDAR scan layer and toggles is depending on the state of the pressed button. Afterwards, the same action is performed on the BAG building geometry, however, when a specific neighborhood is selected, only the building geometry within that neighborhood is shown.

B.3 toggleNeighborhoods()

Dependencies

updatePostalCodes(emptyResult) B.11 (p. 71)

Code

```
1: if neighborhoods need to be disabled then
2:     for all postal code geometries do
3:         hide geometry
4:     end for
5: else
6:     updatePostalCodes(emptyResult)
7: end if
```

B.4 toggleAll(show)

Input Variables

- show - expected to be a string value indicating the box that is toggled and therefore needs to be shown or hidden.

Code

```
1: if the area to be toggled is part of areas that can be toggled then
2:   for all areas that can be toggled do
3:     if the area to be toggled is this area then
4:       if the area to be toggled is already shown then
5:         hide area to be toggled
6:         make map full width of screen
7:       else
8:         show the area to be toggled
9:       end if
10:    end if
11:  end for
12: else
13:  for all areas that can be toggled do
14:    hide area
15:    make map full width of screen
16:  end for
17: end if
```

B.5 updateTable(postalCode)

Input Variables

- postalCode - expected to be a string in the format '0000AA' representing a 6-digit postal code as defined on page 17. This variable represents the fact that a single postal code is selected.

Code

```
1: if valuesDict contains any variables then
2:   for all variables in valuesDict do
3:     add a column to the table
4:   end for
5:   if postalCode is false then
6:     for all levels of postal code (4,5,6) do
7:       for variable in valuesDict do
8:         for postal code in valuesDict[variable] do
9:           add row to table
10:        end for
11:      end for
12:    end for
13:  else
14:    only for the selected postal code
15:    for all levels of postal code (4,5,6) do
16:      for variable in valuesDict do
17:        for neighborhood in valuesDict[variable] do
18:          add row to table
19:        end for
20:      end for
21:    end for
22:  end if
23: else
24:   hide results table
25: end if
```

B.6 orderTable(variable, ordering)

Input Variables

- variable - expected to be a string representing the relevant variable
- ordering - expected to be a string variable, either 'highToLow', 'lowTohigh' or 'none'. Reflects the way the column needs to be ordered.

Auxilliary Variables

- createTableDict - dictionary created in function B.6 which is required for the creation of the data table. Is in the format:

```
{
        postal code :
                {
                        variable : value
                }
}
```

Dependencies

- updateTable(postalCode) B.5 (p. 65)

Code

```
1: orderedArray = []
2: for postal code in createTableDict do
3:    set value as the appropriate value associated with this postalCode
4:    for all values in orderedArray do
5:       if ordering is 'highToLow' then
6:          if value is larger than value in orderedArray then
7:             insert value and corresponding postal code before the current
               value of orderedArray
8:          else
9:             add value and corresponding postal code to the end of or-
               deredArray
10:         end if
11:      else if ordering is 'lowToHigh' then
12:         if value smaller than value in orderedArray then
13:            insert value and corresponding postal code before the current
               value of orderedArray
14:         else
15:            add value and corresponding postal code to the end of or-
               deredArray
16:         end if
17:      end if
18:   end for
19:   if ordering is 'none' then
20:      updateTable(false)
21:   else
22:      for all rows in orderedArray do
23:         for all variables do
24:            add row to table
25:         end for
26:      end for
27:   end if
28: end for
```

This function hides all other available 'boxes' which could show on the right side of the dashboard, while showing the one currently toggled (or the reverse). Moreover, it halfs the display size of the map in order to make room for the information. The toggle direction (show or hide) is based on the status of the pressed button.

B.7 getMetaData()

Query

- C.2 (p. 105)

Code

```
1: query relevant meta data
2: for all results of query do
3:    if result is a property or a class then
4:       add as a 'base' node
5:    else
6:       add as a 'normal' node
7:    end if
8:    add edge between correct nodes
9: end for
10: create graph with base nodes and edges
11: on click:
12: if clicked node is previously clicked node then
13:    for connected edges do
14:       if connected edges and nodes are not connected to other (still visi-
          ble) nodes or edges then
15:          remove connected edges and nodes
16:       end if
17:    end for
18: else
19:    add edges and nodes that are connected to the clicked node to the
       graph
20: end if
```

This function defines the creation of the default network graph (Section 3.4). First, relevant data is queried from the graph database, after which, for each line in the result a node and edge are created. Only the 'base' of the graph is visualized (see Figure 11). In the creation of the graph, also the interactive behavior is defined. Where the graph is expanded when the user clicks on a node. When an already expanded node is clicked again, the connected edges and nodes are again hidden, if they are not also connected to another visible node or edge.

B.8 getInstanceData(postalCode)

Input Variables

- postalCode - expected to be a string in the format '0000AA' representing a 6-digit postal code as defined on page 17. This variable represents the fact that a single postal code is selected.

Query

- C.3 (p. 106)

Code

```
query properties of postal code
for all spatial levels do
    add nodes (from query result) to the node of this spatial level
    add edges (from query result) between nodes at this spatial level
    add edges between relevant spatial levels
end for
create graph
on click:
if clicked node is previously clicked node then
    for connected edges do
        if connected edges and nodes are not connected to other (still visible) nodes or edges then
            remove connected edges and nodes
        end if
    end for
else
    add edges and nodes that are connected to the clicked node to the graph
end if
```

This function is comparable to function B.8, however, as the query is based on a selected postal code for this function. Moreover, as properties of a postal code can be inherited from containing neighborhoods (see Section 2.1), those neighborhoods and connected properties are added to the graph. These nodes are connected to the selected postal code in the graph.

B.9 togglePC(postalCode, building)

Input Variables

- postalCode - expected to be a string in the format '0000AA' representing a 6-digit postal code as defined on page 17. This variable represents the fact that a single postal code is selected.
- building - expected to be a string or boolean indicating the relevant building URI, alternatively the variable can also be declared False.

Dependencies

• visualizeScore(variableName, showArea)	B.13	(p. 73)
• loadInfo(postalCode)	B.12	(p. 71)
• updateTable(postalCode)	B.5	(p. 65)
• toggleAll(show)	B.4	(p. 64)

Auxilliary Variables

- scoredVariable - global variable indicating which variable is currently being visualized on the map (function B.13). Is False by default, and when no variable is being visualized.

Code

```
1: if a variable is visualized then
2:     visualizeScore(scoredVariable, true)
3: end if
4: if the clicked element is not a building (i.e. building is false) then
5:     if the clicked element is not the previously clicked postal code geom-
       etry then
6:         highlight the clicked postal code geometry
7:         zoom to the clicked postal code
8:     else
9:         toggleInfoBox()
10:        remove highlight from the clicked postal code
11:    end if
12: else
13:    loadInfo(postalCode)
14: end if
```

This function determines what action to perform when a geometry is clicked in the map. If it is a postal code, it might be the first time that that postal code is clicked, in which case the postal code will be highlighted. If the postal code has already been clicked, the highlight will be removed. When the clicked element is a building, function B.12 is called.

B.10 predictEnergyLabel(buildingNumber, postalCode, buildYear, area)

Input Variables

- buildingNumber - expected to be a string indicating the BAG URI for the specific building.
- postalCode - expected to be a string in the format '0000AA' representing a 6-digit postal code as defined on page 17. This variable represents the fact that a single postal code is selected.
- buildYear - expected to be an integer reflecting the year the building was build, in the format 2023.
- area - expected to be an integer reflecting the area of the building in square meters, in the format 50.

Query

- C.4 (p. 107)

Code

```
query relevant data for selected building
if building has energy label then
    return energy label
else
    if all necessary data for Random Forest is available then
        use found data to make API call to Random Forest model
        return 'C or better' or 'Worse than C' according to API response
    else
        return 'Could not be determined'
    end if
end if
```

B.11 updatePostalCodes(emptyResult)

Input Variables

- emptyResult - boolean indicating if another function has found that no results where determined using the existing query, or if the map needs to be reset to the original visualization. Note: code below will be run to check if this boolean is set correctly.

Dependencies

- updateTable(postalCode) B.5 (p. 65)

Code

```
1: for all variables in postalCodeDict do
2:   if variable has no postal codes associated with it then
3:     label variable as 'empty'
4:   end if
5: end for
6: if all variables are empty then
7:   reset the visualization of all postal code geometries
8:   notify user with warning that no results where found
9: else if not all variables are empty then
10:  updateTable(false)
11:  for all postal code geometries do
12:    for all variables in postalCodeDict do
13:      if variable includes this postal code and is not included in the
         nanDict then
14:        label postal code as 'included'
15:      else if postal code is included in the nanDict then
16:        label postal code as 'nan'
17:      else
18:        label postal code as 'not included'
19:      end if
20:    end for
21:    if postal code is labeled as 'not included' then
22:      hide postal code geometry on map
23:    else if postal code is labeled as 'nan' then
24:      show postal code geometry as being unknown
25:    else
26:      show postal code geometry
27:    end if
28:  end for
29:  zoom to shown postal code geometries
30: end if
```

B.12 loadInfo(postalCode)

Input Variables

- postalCode - expected to be a string in the format '0000AA' representing a 6-digit postal code as defined on page 17. This variable represents the fact that a single postal code is selected.

Query

- C.5 (p. 108)

Code

```
1: query Cadastre for all buildings in the selected postal code
2: filter BAG geometries to only load the found buildings
3: query graph database for all relevant data for the selected postal code
4: create table based on found information
```

B.13 visualizeScore(variable, showArea)

Input Variables

- variable - expected to be a string representing the relevant variable
- showArea - expected to be a boolean which indicates whether a specific neighborhood is selected, which needs to be visualized.

Dependencies

- updatePostalCodes(emptyResult) B.11 (p. 71)

Code

```
1: if variable is not the variable that is already visualized or a postal code
   is selected then
2:    determine minimum and maximal values of scored variable based on
      already shown postal code geometries
3:    for all postal code geometries do
4:       if variable is unknown for postal code then
5:          show postal code geometry as unknown
6:       else
7:          show postal code geometry in correct color according to value of
            the variable for that postal code, and minimum and maximum
            values of scored variable
8:       end if
9:    end for
10:   zoom to shown postal code geometries
11: else
12:   updatePostalCodes(false)
13: end if
```

B.14 resetView()

Dependencies

- removeVariable(variable) B.21 (p. 81)
- createEPGraph(toggle, epType, prediction) B.30 (p. 90)
- updatePostalCodes(emptyResult) B.11 (p. 71)

Code

```
1: for all variables in postalCodeDict do
2:     removeVariable(variable)
3: end for
4: reset postalCodeDict
5: if the energy poverty analysis has already been done at least once in the
   session then
6:     createEPGraph(false, 'slider')
7: else
8:     updatePostalCodes(true)
9: end if
```

B.15 showArea(postalCode)

Input Variables

- postalCode - expected to be a string in the format '0000AA' representing a 6-digit postal code as defined on page 17. This variable represents the fact that a single postal code is selected.

Dependencies

- visualizeScore(variable, showArea) B.13 (p. 73)

Query

C.6 (p. 109)
C.7 (p. 110)
C.8 (p. 111)
C.9 (p. 112)

Code

```
1: get selected spatial level from slider
2: construct query based on spatial level
3: if a scored variable is defined then
4:     visualizeScore(scoredVariable, true)
5: end if
6: for all postal code geometries do
7:     if postal code is not selected then
8:         if postal code geometry is already shown and no variable is scored
           then
9:             show postal code geometry regularly
10:        else if a variable is scored then
11:            visualizeScore(scoredVariable, true)
12:        end if
13:    else
14:        show postal code geometry highlighted
15:    end if
16: end for
17: only show data that corresponds to the selected spatial level in the
    information table
```

B.16 loadBuildingEnergyUse(building)

Input Variables

- building - expected to be a string or boolean indicating the relevant building URI, alternatively the variable can also be declared False.

Query

- C.10 (p. 113)

Code

```
query correct database and ID
if a database was found then
    query database using ID
    create graph
else if selected building was already selected then
    hide already created graph
else
    notify user that no database was found
end if
```

B.17 getQueryVariables(energy)

Input Variables

- energy - boolean indicating whether the selected dropdown menu is related to energy (true) or regular (false) query variables.

Query

- C.11 (p. 114)
- C.12 (p. 115)

Code

```
1: if dropdown is related to energy variables then
2:    empty previous dropdown options
3:    query all energy variables from graph database
4:    if energy variable from query variable is not already selected then
5:       add energy variable to dropdown as recommended
6:    end if
7:    query all regular variables from graph database
8:    if variable from query is not already selected then
9:       add variable to dropdown
10:   end if
11: else
12:   empty previous dropdown options
13:   query all regular variables from graph database
14:   if variable from query is not already selected then
15:      add variable to dropdown
16:   end if
17: end if
```

B.18 updateRatioSlider(variable, minMax, inputType, procedureName)

Input Variables

- variable - expected to be a string representing the relevant variable minMax - expected to be a string indicating whether the input type relates to the minimal or maximal value (expected to be either 'min' or 'max')
- inputType - expected to be a string indicating whether the input type is 'slider' or 'text', allowing the function to be used by both the sliders and the text input.
- procedureName - expected to be a string indicting the procedure of the variable.

Dependencies

- updateRatioValue(variable, inputType) B.20 (p. 78)
- updatePostalCodes(emptyResult) B.11 (p. 71)
- visualizeScore(variable, showArea) B.13 (p. 73)
- createEPGraph(toggle, epType, prediction) B.30 (p. 90)

Query

C.13 (p. 116)

Code

```
1: if minimal value is higher than maximal value then
2:     adjust the maximal value to 30% higher than the minimal value
3: else if maximial value is lower than minimal value then
4:     adjust the minimal value to be 30% lower than the maximal value
5: end if
6: query all neighborhoods and their associated values for this variable and
   procedure within the selected time frame
7: determine the maximal and minimal values based on the query
8: if value value falls within the selected range then
9:     add postal code to the postalCodeDict
10:    add postal code to the valuesDict for this variable
11: end if
12: set the minimal and maximal values of the sliders and text inputs
13: if the energy poverty analysis has been run before in this session then
14:    createEPGraph(false, 'slider'
15: else
16:    if no values where found then
17:        updatePostalCodes(true)
18:    else if a variable is scored then
19:        visualizeScore(scoredVariable, true
20:    else
21:        updatePostalCode(false)
22:    end if
23: end if
```

B.19 updateDateSlider(variable)

Input Variables

- variable - expected to be a string representing the relevant variable

Code

```
1: get selected date from slider
2: update the text associated with the slider
3: set the selected date globally as the date to use for other functions
```

B.20 updateRatioValue(variable, inputType)

Input Variables

- variable - expected to be a string representing the relevant variable
- inputType - expected to be a string indicating whether the input type is 'slider' or 'text', allowing the function to be used by both the sliders and the text input.

Code

```
1: determine minimum and maximum values
2: adjust minimum and maximum of slider and text inputs for this variable
   according to the previously found values
```

B.21 removeVariable(variable)

Input Variables

- variable - expected to be a string representing the relevant variable

Dependencies

- updatePostalCodes(emptyResult) B.11 (p. 71)

Code

1: delete variable from the *postalCodeDict*
2: delete variable from the *valueDict*
3: delete variable from the *nanDict*
4: *updatePostalCodes(**true**)*

B.22 onButtonSelect(variable, category, procedureName)

Input Variables

- variable - expected to be a string representing the relevant variable
- category - expected to be a string representing the selected category
- procedureName - expected to be a string indicting the procedure of the variable.

Dependencies

- updatePostalCodes(emptyResult) B.11 (p. 71)

Query

- C.14 (p. 117)

Code

```
1: if clicked has already been clicked then
2:    label the category associated with the clicked button as no longer
      relevant
3: else
4:    label the category associated with the clicked button as relevant
5: end if
6: if there are no relevant categories then
7:    set postalCodeDict to none for this variable
8:    updatePostalCode(false)
9: else
10:   for all relevant categories do
11:      query all neighborhoods for this category within the selected time-
         frame
12:      add the queried neighborhoods to the postalCodeDict for this vari-
         able
13:      updatePostalCodes(false)
14:   end for
15: end if
```

B.23 showSinglePostalCode(postalCode)

Input Variables

- postalCode - expected to be a string in the format '0000AA' representing a 6-digit postal code as defined on page 17. This variable represents the fact that a single postal code is selected.

Dependencies

- updatePostalCodes(emptyResult) B.11 (p. 71)

Code

```
1: if the selected postal code is also the last selected postal code then
2:    state globally that there is no last selected postal code
3:    updatePostalCodes(false)
4: else
5:    for all postal code geometries do
6:       if postal code geometry is the selected postal code then
7:          show postal code geometry
8:       else
9:          hide postal code geometry
10:      end if
11:   end for
12:   state globally variable that the selected postal code is the last selected
      postal code
13: end if
```

B.24 fullScreen()

Code

```
1: if graph and table are half width then
2:     set graph and table to full width
3: else
4:     set graph and table to half width
5: end if
```

B.25 showResults()

Code

```
1: if table with results is already shown then
2:     hide table with results
3: else
4:     show table with results
5: end if
```

B.26 exportData()

Code

```
1: transform results table to JSON format
2: transform JSON to .csv format
3: create a download link, which links to the .csv document
4: automatically click the download link
```

B.27 showGraph(graphType)

Input Variables

- graphType - expected to be a string indicating if the meta data ('data') or neighborhood specific ('instance) graph needs to be shown

Dependencies

- getMetaData() B.7 (p. 67)
- getInstanceData(postalCode) B.8 (p. 68)

Code

```
1: if metadata needs to be shown and no graph has been shown last or
   metadata needs to be shown and the last shown graph has been the
   neighborhood graph then
2:     show the general graph space
3:     getMetaData()
4: else if neighborhood data needs to be shown and no graph has been
   shown last or neighborhood data needs to be shown and the last shown
   graph has been the metadata graph then
5:     show the general graph space
6:     getInstanceData(selectedPostalCode)
7: else
8:     hide the general graph space
9: end if
```

B.28 getTimeSeries(level)

Input Variables

- level - expected to be a string which indicates which level of analysis is requested, can be either 'pc' for neighborhoods, 'building' for building level or 'both' for a combination

Dependencies

- updatePostalCodes(emptyResult) B.11 (p. 71)
- visualizeScore(variable, showArea) B.13 (p. 73)
- createTimeSeriesDict() B.32 (p. 94)

Query

C.15 (p. 118)

Code

```
1: if level is neighborhood then
2:    query the most recent electricity use on a postal code level
3:    update the postalCodeDict
4:    updatePostalCodes(false)
5:    visualizeScore('electricityUse', true)
6: else if level is building then
7:    if time series data has not yet been created on a building level then
8:       createTimeSeriesDict()
9:    end if
10:   for all neighborhoods including buildings with time series electricity
      use data do
11:      update nanDict using building data
12:      update postalCodeDict using building data
13:   end for
14:   updatePostalCodes(false)
15:   visualizeScore('electricityUse, true)
16: else if level is both then
17:   if time series data has not yet been created on a building level then
18:      createTimeSeriesDict()
19:   end if
20:   for all neighborhoods including buildings with time series electricity
      use data do
21:      update nanDict using both neighborhood and building data
22:      update postalCodeDict using both neighborhood and building data
23:   end for
24:   updatePostalCodes(false)
25:   visualizeScore('electricityUse, true)
26: end if
```

B.29 getRTBuildings()

Dependencies

- queryDBrealTime(id, params) B.33 (p. 95)

Query

- C.16, (p. 119)

Code

1: query all buildings with an associated database
2: clear dropdown menu of previous options
3: **for all** found buildings **do**
4: add building as option to dropdown menu with interactive function *queryDBrealTime(id, params)*
5: **end for**

B.30 createEPGraph(toggle, epType, prediction)

Input Variables

- toggle - boolean indicating whether the energy poverty space needs to be toggled.
- epType - expected to be a string indicating the which definition of energy poverty needs to be visualized ('10percent, 'lihe' or 'lilek').
- prediction - boolean indicating whether predicted energy labels are to be included in the analysis

Dependencies

- toggleAll('energyPovertyBox') B.4 (p. 64)
- updateTable(postalCode) B.5 (p. 65)
- updatePostalCodes(emptyResults) B.11 (p. 71)
- createEPData() B.34 (p. 96)
- createLILEKdata() B.35 (p. 97)
- createBuildingDict(data, prediction) B.36 (p. 98)

Code

```
1: if energy poverty data has not yet been collected then
2:    createEPdata()
3: end if
4: if energy poverty definition is '10percent' then
5:    establish the median income based on the found data
6:    establish median energy cost based on the found data
7:    for all found neighborhoods do
8:       if income is not defined for this neighborhood then
9:          use median income
10:      end if
11:      establish the energy cost for this neighborhood
12:      if the number of households for this neighborhood is known then
13:         if neighborhood can be considered to be at risk of energy poverty
            according to the correct poverty line based on the amount of
            households and current definition then
14:            mark the neighborhood to be included in the graph as at risk
               of energy poverty.
15:         else if neighborhood is not considered to be at risk of energy
            poverty but is added or subtracted according to the adjusted
            definition of energy poverty then
16:            mark neighborhood to be included as added or subtracted
17:         else
18:            mark neighborhood as not at risk of energy poverty
19:         end if
20:      end if
21:   end for
22:   update information based on the found results
23: else if energy poverty definition is 'lihe' then
24:   establish the median income based on the found data
25:   establish median energy cost based on the found data
26:   for all found neighborhoods do
27:      if income is not defined for this neighborhood then
28:         use median income
29:      end if
30:      establish the energy cost for this neighborhood
```

```
31:       if the number of households for this neighborhood is known then
32:         if neighborhood can be considered to be at risk of energy poverty
            according to the correct poverty line based on the amount of
            households and current definition then
33:           mark the neighborhood to be included in the graph as at risk
              of energy poverty.
34:         else if neighborhood is not considered to be at risk of energy
            poverty but is added or subtracted according to the adjusted
            definition of energy poverty then
35:           mark neighborhood to be included as added or subtracted
36:         else
37:           mark neighborhood as not at risk of energy poverty
38:         end if
39:       end if
40:     end for
41:     update information based on the found results
42:   else if energy poverty definition is 'lilek' then
43:     if data was not yet collected on a building level then
44:       createLILEKdata()
45:       createBuildingDict(data, prediction)
46:     else
47:       createBuildingDict(data, prediction)
48:     end if
49:     for all found neighborhoods do
50:       establish the median income based on the found data
51:       establish median energy cost based on the found data
52:       if energy labels are present in the neighborhood which are below C
          then
53:         if neighborhood can be considered to be at risk of energy poverty
            according to the correct poverty line based on the number of
            households and current definition then
54:           mark the neighborhood to be included in the graph as at risk
              of energy poverty.
55:         else if neighborhood is not considered to be at risk of energy
            poverty but is added or subtracted according to the adjusted
            definition of energy poverty then
56:           mark neighborhood to be included as added or subtracted
57:         else
58:           mark neighborhood as not at risk of energy poverty
59:         end if
60:       end if
61:       update information based on the found results
62:     end for
63:   end if
64:   create graph based on the found data
65:   add interactivity to the graph:
66:   if energy poverty definition is not 'lilek' then
67:     if clicked dot was clicked previously then
68:       remove highlight from dot
69:       update postalCodeDict for the 'energy poverty' variable
70:       updatePostalCodes(false)
71:     else
72:       highlight dot
73:       update postalCodeDict for the 'energy poverty' variable
74:       updatePostalCodes(false)
75:     end if
76:     updateTable(postalCode)
77:   else
78:     if clicked dot was clicked previously then
```

```
79:       remove highlight from dot
80:     else
81:       highlight dot
82:     end if
83:     create pie chart based on energy labels that were previously found for
        the clicked dot's postal code
84:     add interactivity to pie chart:
85:     if energy label is not the last selected energy label then
86:       highlight the buildings associated with the selected energy label
87:     else
88:       show LiDAR scan
89:     end if
90: end if
91: update postalCodeDict for 'energy poverty' variable
92: updateTable(false)
93: updatePostalCodes(false)
94: if energy poverty space needs to be toggled then
95:   toggleEPBox()
96: end if
```

B.31 updateAPLText(energy)

Input Variables

- energy - expected to be a string which reflects if an energy variable is selected.

Code

```
get the margin by which the energy poverty definition is adjusted
if check if the selection type refers to an energy variable then
    change the displayed text to include the adjusted margin
end if
```

B.32 createTimeSeriesDict()

Dependencies

- queryDB(buildingData) B.37 (p. 99)

Query

- C.17 (p. 120)
- C.18 (p. 121)

Code

```
1: query electricity use on a neighborhood level from graph database
2: for all results from query do
3:     store the found electricity use as the average for the neighborhood
4: end for
5: query all buildings with an associated external database
6: for all results from query do
7:     add the result from queryDB(result) to the total electricity use on a
       building level to the stored total found in the neighborhood
8:     add 1 to the total number of buildings in the area
9: end for
```

B.33 queryDBrealTime(id, params)

Input Variables

- id - expected to be a string which can be used to make the API call, pointing towards to correct database.
- params - expected to be a dictionary with keys indicating a parameter and a corresponding value of that parameter.

Dependencies

- toggleRt() B.38 (p. 100)
- createRTGraph() B.39 (p. 101)

Code

```
1: for all parameters in params do
2:     add parameter to id (URL) for API call
3: end for
4: retrieve data by making API call using previously created API URL
5: toggleRT
6: createRTGraph()
```

B.34 createEPdata()

Query

- C.19 (p. 122)
- C.20 (p. 123)
- C.21 (p. 124)

Code

```
1: query dwellings and total amount of buildings for all neighborhoods
2: query energy use, income and average building value for all neighbor-
   hoods
3: query number and composition of households for all neighborhoods
4: store retrieved data
```

B.35 createLILEKdata()

Query

- C.22 (p. 125)

Code

```
1: query all buildings and their (predicted) energy label, build year and
   area
2: return found data
```

B.36 createBuildingDict(data, prediction)

Input Variables

- data - expected to be a dictionary with data created in function B.38
- prediction - expected to be a boolean indicating whether predicted energy labels should be included.

Code

```
for all buildings in the data do
  if predictions need to be included (prediction is true) then
    if building energy label is worse than C then
      store building as 'energy poor'
      store building energy label
    else if building energy label is better or equal to C then
      store building as 'not energy poor'
    else if building energy label is unknown then
      store building as 'unknown'
    end if
  else
    if building energy label measurement procedure is marked as the
    actual energy label (not predicted) then
      if building energy label is worse than C then
        store building as 'energy poor'
        store building energy label
      else if building energy label is better or equal to C then
        store building as 'not energy poor'
      else if building energy label is unknown then
        store building as 'unknown'
      end if
    end if
  end if
end for
return stored data
```

B.37 queryDB(buildingData)

Input Variables

- buildingData - expected to be data including a database ID which can be called in an API call, created from function B.35.

Code

```
if building data includes correctly formatted query ID (API URL) then
    make API call
    return value returned by API call
else
    return value is unknown
end if
```

B.38 toggleRt()

Code

1: show real-time energy use space
2: update text to reflect found current energy use

B.39 createRTGraph()

Dependencies

- getWeatherData(uniqueDates) B.40 (p. 102)

Code

```
1: for all values found in the time series database for the selected building
   do
2:    add value and date to the graph
3:    if date has not been recorded before then
4:       store date as a new unique date
5:    end if
6: end for
7: getWeatherData(uniqueDates
8: for all data points found for the weather do
9:    add value and date to weather graph
10: end for
11: create graph using time-series energy use and weather
```

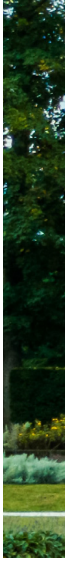
B.43 getWeatherData(uniqueDates)

Input Variables

- uniqueDates - expected to be a list of dates created in function B.39.

Code

```
for all unique dates do
    make API call
    for all hours of that day do
        store temperature for hour of the unique date
    end for
end for
return  found temperature for each hour of each unique date
```



Appendix C Queries

This appendix provides an overview of all the SPARQL queries used in the digital twin as described in this report. Appendix B denotes which query is used in which aspect of the code, it is therefore recommended to assess these queries in the context of the code they are a part of.

Moreover, most of the queries are posted to the locally run graph database, the endpoint of which is denoted in this appendix as: 'http://localhost:7200/repositories/[REPOSITORY]'. The repository name is left empty as the repository can name is arbitrary. If the query is posted to any other (external) endpoint, this will be noted in above the query description.

C.1

Endpoint: https://api.labs.kadaster.nl/datasets/dst/kkg/services/default/sparql

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX bag: <http://bag.basisregistraties.overheid.nl/def/bag#>
PREFIX bag_shp: <http://bag.basisregistraties.overheid.nl/bag/id/shape/>
SELECT ?pand ?adres ?postcode ?huisnummer ?toevoeging ?oppervlakte
        ?bouwjaar ?gebruiksdoel WHERE {
        ?verblijfsobject a bag:Verblijfsobject.
        ?verblijfsobject bag:maaktDeelUitVan ?pand.
        FILTER(?pand = pandURI)
        ?verblijfsobject bag:hoofdadres ?adres.
        ?verblijfsobject bag:gebruiksdoel ?gebruiksdoel.
        ?adres bag:postcode ?postcode.
        ?adres bag:huisnummer ?huisnummer.
        OPTIONAL {
                ?adres bag:huisnummertoevoeging ?toevoeging
        }
        ?verblijfsobject bag:oppervlakte ?oppervlakte.
        ?pand bag:oorspronkelijkBouwjaar ?bouwjaar.
}
```

C.2

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
select distinct ?propertyName ?LOM ?pcLevel ?class ?procedureName where {
        ?property a bop:Property.
        ?property skos:prefLabel ?propertyName.
        ?property a ?class.
        FILTER (STRSTARTS(str(?class), 'http://www.semanticweb.org/neighborhood-
        energy-ontology#') && str(?class) != 'http://www.semanticweb.org/
        neighborhood-energy-ontology#Gasuse' && str(?class) != 'http://www.
        semanticweb.org/neighborhood-energy-ontology#Electricityuse')
        ?property bop:hasExecution ?execution.
        ?execution neo:levelOfMeasurement ?LOM.
        ?execution bop:usesProcedure ?procedure.
        ?procedure skos:prefLabel ?procedureName.

        ?neighborhood bop:hasProperty ?property.
        ?neighborhood bag2:postcode ?postcode.
        BIND (strlen(str(?postcode)) AS ?pcLevel).
} limit 100
```

C.3

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
select distinct ?postcode ?propertyName ?procedureName ?pcLevel ?lom ?unit
?definition where {
        ?neighborhood a neo:Neighborhood.
        ?neighborhood bag2:postcode ?postcode.
        FILTER((STRSTARTS(STR(?postcode), 'postalCode')) || ?postcode =
        'postalCode' || ?postcode = 'postalCode.substring(0, 5)' || ?postcode =
        'postalCode.substring(0, 4)')

        BIND(strlen(STR(?postcode)) AS ?pcLevel)
        ?neighborhood bop:hasProperty ?property.
        ?property skos:prefLabel ?propertyName.

        ?property bop:hasExecution ?execution.
        ?execution neo:levelOfMeasurement ?lom.
        ?execution skos:definition ?definition.
        ?execution bop:hasResult/bop:hasSimpleUnit ?unit.

        ?execution bop:usesProcedure ?procedure.
        ?procedure skos:prefLabel ?procedureName.
} limit 100
```

C.4

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
PREFIX bot: <https://w3c-lbd-cg.github.io/bot/#>
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX seas: <https://w3id.org/seas/EvaluationOntology#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
select distinct ?pc ?building ?energyLabel ?propertyName ?value where {
        ?pc a neo:Neighborhood.
        ?pc bag2:postcode ?postcode.
        FILTER(?postcode = 'postalCode')
        ?pc ^bot:containsZone+ ?neighborhood.
        {
                ?neighborhood bop:hasProperty ?property.
        } UNION {
                ?pc bop:hasProperty ?property.
        }
        ?pc bot:containsZone ?building.
        FILTER(strStarts(str(?building), 'https://bag2.basisregistraties.overheid.nl/bag/
        id/registratie/NL.IMBAG.Nummeraanduiding.nummeraanduiding'))
        OPTIONAL {
                ?building bop:hasProperty ?elProperty.
                ?elProperty bop:hasValue ?energyLabel.
        }
        FILTER(?propertyName in ('incomePerRecipient', 'electricityUse', 'gasUse',
        'WOZValue', 'powerGeneration'))
        ?property skos:prefLabel ?propertyName.
        ?property bop:hasExecution ?execution.
        ?execution bop:hasResult ?result.
        ?result seas:hasTemporalContext ?context.
        ?context time:hasBeginning/time:inXSDDateTimeStamp ?begin.
        FILTER (?begin = '2020-01-01T00:00:00'^^xsd:dateTimeStamp)
        ?result bop:hasValue ?value.
}
```

C.5

Endpoint: https://api.labs.kadaster.nl/datasets/dst/kkg/services/default/sparql

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX bag: <http://bag.basisregistraties.overheid.nl/def/bag#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select ?Pand where {
        ?nummeraanduiding a bag:Nummeraanduiding.
        ?nummeraanduiding bag:postcode ‘postalCode’.
        ?verblijfsobject bag:hoofdadres ?nummeraanduiding.
        ?verblijfsobject bag:maaktDeelUitVan ?Pand.
}
```

C.6

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
select ?property ?postcode ?otherPostcode where {
        ?neighborhood bag2:postcode ?postcode
        FILTER(?postcode = 'postalCode)
        ?neighborhood bop:hasProperty ?property.
        ?property a neo:ElectricityuseProperty.
        ?otherNeighborhood bop:hasProperty ?property.
        ?otherNeighborhood bag2:postcode ?otherPostcode.
}
```

C.7

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
select ?property ?postcode ?otherPostcode where {
        ?neighborhood bag2:postcode ?postcode
        FILTER(?postcode = 'postalCode')
        ?neighborhood bop:hasProperty ?property.
        ?property a neo:GasuseProperty.
        ?otherNeighborhood bop:hasProperty ?property.
        ?otherNeighborhood bag2:postcode ?otherPostcode.
}
```

C.8

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
select distinct ?postcode where {
        ?neighborhood a neo:Neighborhood.
        ?neighborhood bag2:postcode ?postcode.
        FILTER(strstarts(str(?postcode), 'postalCode'))
}
```

C.9

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
PREFIX bot: <https://w3c-lbd-cg.github.io/bot/#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
select distinct ?neighborhoodName ?postcode where {
        ?postalCodeArea a neo:Neighborhood.
        ?postalCodeArea bag2:postcode ?selectedPostalCode.
        FILTER(?selectedPostalCode = 'postalCode')
        ?neighborhood bot:containsZone ?postalCodeArea.
        ?neighborhood skos:prefLabel ?neighborhoodName.
        ?neighborhood bot:containsZone ?otherPostalCodeAreas.
        ?otherPostalCodeAreas bag2:postcode ?postcode
}
```

C.10

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
select ?building ?id where {
        ?building bop:hasProperty ?electricityUseProperty.
        FILTER(?building = <buildingURI>)
        ?electricityUseProperty bop:hasPropertyState ?dataPoint.
        ?dataPoint bop:hasID ?id.
} limit 100
```

C.11

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
select distinct ?propertyName ?definition ?unit ?procedureName where {
        ?property a neo:EnergyUse.
        ?property skos:prefLabel ?propertyName.
        ?property bop:hasExecution ?execution.
        ?execution skos:definition ?definition.
        ?execution bop:usesProcedure ?procedure.
        ?procedure skos:prefLabel ?procedureName.

        ?execution bop:hasResult/bop:hasSimpleUnit ?unit.
} limit 100
```

C.12

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
select distinct ?propertyName ?definition ?unit ?procedureName where {
        ?property a bop:Property.
        FILTER NOT EXISTS {?property a neo:EnergyUse}
        ?property skos:prefLabel ?propertyName.
        ?property bop:hasExecution ?execution.
        ?execution skos:definition ?definition.
        ?execution bop:usesProcedure ?procedure.
        ?procedure skos:prefLabel ?procedureName.

        ?execution bop:hasResult/bop:hasSimpleUnit ?unit.
} limit 100
```

C.13

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX seas: <https://w3id.org/seas/EvaluationOntology#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX bot: <https://w3c-lbd-cg.github.io/bot/#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
select distinct ?postcode ?value where {
        {
                ?neighborhood a neo:Neighborhood.
                ?neighborhood bag2:postcode ?postcode.
                ?neighborhood bop:hasProperty ?property.
        } UNION {
                ?neighborhood a neo:Neighborhood.
                ?neighborhood skos:prefLabel ?postalCode.
                FILTER(strstarts(str(?postalCode), 'BU'))
                ?neighborhood bot:containsZone ?zone.
                ?zone bag2:postcode ?postcode .
                ?neighborhood bop:hasProperty ?property.
        }
        ?property skos:prefLabel ?propertyName.
        FILTER(?propertyName = 'variable')
        ?property bop:hasExecution ?execution.
        ?execution bop:usesProcedure ?procedure.
        ?procedure skos:prefLabel ?procedureName.
        FILTER(?procedureName = 'procedure')
        ?execution bop:hasResult ?result.
        ?result seas:hasTemporalContext ?interval.
        ?interval time:hasBeginning/time:inXSDDateTimeStamp ?begin.
        ?interval time:hasEnd/time:inXSDDateTimeStamp ?end.
        FILTER(?begin >= "beginDate"^^xsd:dateTimeStamp && ?end <=
        "endDate"^^xsd:dateTimeStamp)
        ?result bop:hasValue ?value
}
```

C.14

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX seas: <https://w3id.org/seas/EvaluationOntology#>
PREFIX time: <http://www.w3.org/2006/time#>
select ?postcode ?value ?begin ?end where {
        ?neighborhood a neo:Neighborhood.
        ?neighborhood bag2:postcode ?postcode.

        ?neighborhood bop:hasProperty ?property.
        ?property skos:prefLabel ?propertyName.
        FILTER(?propertyName = 'variable')
        ?property bop:hasExecution ?execution.
        ?execution bop:usesProcedure ?procedure.
        ?procedure skos:prefLabel ?procedureName.
        FILTER(?procedureName = 'procedure')
        ?execution bop:hasResult/bop:hasValue ?value.
        FILTER(?value = category || ?value = 'category')

        ?execution bop:hasResult ?result.
        ?result seas:hasTemporalContext ?interval.
        ?interval time:hasBeginning/time:inXSDDateTimeStamp ?begin.
        ?interval time:hasEnd/time:inXSDDateTimeStamp ?end.
        FILTER(?begin >= "beginDate"^^xsd:dateTimeStamp && ?end <=
        "endDate"^^xsd:dateTimeStamp)
}
```

C.15

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX seas: <https://w3id.org/seas/EvaluationOntology#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX bot: <https://w3c-lbd-cg.github.io/bot/#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
select distinct ?postcode ?electricityValue where {
        ?neighborhood bot:containsZone ?pcZone.
        ?pcZone bag2:postcode ?postcode.
        ?pcZone bop:hasProperty ?electricityUseProperty.
        ?electricityUseProperty skos:prefLabel ?electricityUseName.
        FILTER(?electricityUseName = 'electricityUse')
        ?electricityUseProperty bop:hasExecution ?electricityExecution.
        ?electricityExecution bop:hasResult ?electricityResult.
        ?electricityResult seas:hasTemporalContext ?electricityTemporalContext.
        ?electricityTemporalContext time:hasBeginning/time:inXSDDateTimeStamp
        ?elecBegin.
        FILTER(?elecBegin = '2022-01-01T00:00:00'^^xsd:dateTimeStamp)
        ?electricityResult bop:hasValue ?electricityValue.
}
```

C.16

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
select distinct ?building ?postcode ?nr ?toevoeging ?dbID ?db ?unit where {
        ?building a bag2:NummeraanduidingRegistratie.
        ?building bag2:postcode ?postcode.”
        ?building bag2:huisnummer ?nr.
        OPTIONAL {
                ?building bag2:huisnummertoevoeging ?toevoeging
        }
        ?building bop:hasProperty ?electricityUse.
        ?electricityUse bop:hasPropertyState ?electricityUseDP.
        ?electricityUseDP bop:isDataPointOf ?db.
        ?electricityUseDP bop:hasID ?dbID.
        ?electricityUseDP bop:hasSimpleUnit ?unit.
}
```

C.17

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX : <https://sanderdemeij.github.io/neo/#>
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
PREFIX bot: <https://w3c-lbd-cg.github.io/bot/#>
PREFIX neo2: <https://sanderdemeij.github.io/neo/#>
PREFIX seas: <https://w3id.org/seas/EvaluationOntology#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
select ?postcode ?electricityValue where {
        ?pcZone bag2:postcode ?postcode.
        ?pcZone bop:hasProperty ?electricityUseProperty.
        ?electricityUseProperty skos:prefLabel ?electricityUseName.
        FILTER(?electricityUseName = 'electricityUse')
        ?electricityUseProperty bop:hasExecution ?electricityExecution.
        ?electricityExecution bop:hasResult ?electricityResult.
        ?electricityResult seas:hasTemporalContext ?electricityTemporalContext.
        ?electricityTemporalContext time:hasBeginning/time:inXSDDateTimeStamp
        ?elecBegin.
        FILTER(?elecBegin = '2022-01-01T00:00:00'^^xsd:dateTimeStamp)
        ?electricityResult bop:hasValue ?electricityValue.
}
```

C.18

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
PREFIX bot: <https://w3c-lbd-cg.github.io/bot/#>
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
select ?postcode ?building ?id where {
        ?building bop:hasProperty ?electricityUseProperty.
        ?electricityUseProperty bop:hasPropertyState ?dataPoint.
        ?dataPoint bop:hasID ?id.
        ?neighborhood bot:containsZone ?building.
        ?neighborhood bag2:postcode ?postcode.
}
```

C.19

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
PREFIX bot: <https://w3c-lbd-cg.github.io/bot/#>
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX seas: <https://w3id.org/seas/EvaluationOntology#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
select distinct ?postcode (sum(if((?gebruiksdoel = 'woonfunctie' || ?gebruiksdoel =
'nan'), 1, 0)) as ?woonBuildings) (count(?building) as ?buildings) where {
        ?building bag2:gebruiksdoel ?gebruiksdoel.
        ?building bag2:postcode ?postcode.
} GROUP BY ?postcode
```

C.20

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX seas: <https://w3id.org/seas/EvaluationOntology#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX bot: <https://w3c-lbd-cg.github.io/bot/#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
select distinct ?neighborhood ?postcode ?value ?electricityValue ?gasValue ?WOZValue where {
        ?neighborhood a neo:Neighborhood.
        ?neighborhood skos:prefLabel ?postalCode.
        FILTER(strstarts(str(?postalCode), 'BU'))

        ?neighborhood bop:hasProperty ?property.
        ?property skos:prefLabel ?propertyName.
        FILTER(?propertyName = 'householdIncome')
        ?property bop:hasExecution ?execution.
        ?execution bop:hasResult ?result.
        ?result bop:hasValue ?value.
        ?result seas:hasTemporalContext ?temporalContext.
        ?temporalContext time:hasBeginning/time:inXSDDateTimeStamp ?begin.
        ?temporalContext time:hasEnd/time:inXSDDateTimeStamp ?end.
        FILTER(?end = '2022-01-01T00:00:00'^^xsd:dateTimeStamp)

        ?neighborhood bot:containsZone ?pcZone.
        ?pcZone bag2:postcode ?postcode.
        ?pcZone bop:hasProperty ?electricityUseProperty.
        ?electricityUseProperty skos:prefLabel ?electricityUseName.
        FILTER(?electricityUseName = 'electricityUse')
        ?electricityUseProperty bop:hasExecution ?electricityExecution.
        ?electricityExecution bop:hasResult ?electricityResult.
        ?electricityResult seas:hasTemporalContext ?electricityTemporalContext.
        ?electricityTemporalContext time:hasBeginning/time:inXSDDateTimeStamp ?elecBegin
        FILTER(?elecBegin = '2022-01-01T00:00:00'^^xsd:dateTimeStamp)
        ?electricityResult bop:hasValue ?electricityValue.

        ?pcZone bop:hasProperty ?gasUseProperty.
        ?gasUseProperty skos:prefLabel ?gasUseName.
        FILTER(?gasUseName = 'gasUse')
        ?gasUseProperty bop:hasExecution ?gasExecution.
        ?gasExecution bop:hasResult ?gasResult.
        ?gasResult seas:hasTemporalContext ?gasTemporalContext.
        ?gasTemporalContext time:hasBeginning/time:inXSDDateTimeStamp ?gasBegin.
        FILTER(?gasBegin = '2022-01-01T00:00:00'^^xsd:dateTimeStamp)
        ?gasResult bop:hasValue ?gasValue.

        ?pcZone bop:hasProperty ?wozProperty.
        ?wozProperty skos:prefLabel ?wozPropertyName.
        FILTER(?wozPropertyName = 'WOZValue')
        ?wozProperty bop:hasExecution ?wozExecution.
        ?wozExecution bop:hasResult ?wozResult.
        ?wozResult seas:hasTemporalContext ?wozTemporalContext.
        ?wozTemporalContext time:hasBeginning/time:inXSDDateTimeStamp ?wozBegin.
        FILTER(?wozBegin = '2020-01-01T00:00:00'^^xsd:dateTimeStamp)
        ?wozResult bop:hasValue ?WOZValue.
}
```

C.21

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX seas: <https://w3id.org/seas/EvaluationOntology#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX bot: <https://w3c-lbd-cg.github.io/bot/#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
select distinct ?postcode ?propertyName ?value where {
        ?pc a neo:Neighborhood.
        ?pc bag2:postcode ?postcode.
        OPTIONAL {
                ?pc bop:hasProperty ?property.
                ?property skos:prefLabel ?propertyName
                FILTER (?propertyName = 'households' || ?propertyName =
                'multPersonHHnoKids' || ?propertyName = 'oneParent' ||
                ?propertyName = 'onePersonHH' || ?propertyName = 'twoParent')

                ?property bop:hasExecution ?execution.
                ?execution bop:hasResult/seas:hasTemporalContext ?temporalContext.
                ?execution bop:hasResult/bop:hasValue ?value.
                ?temporalContext time:hasBeginning/time:inXSDDateTimeStamp
                ?begin.
                ?temporalContext time:hasEnd/time:inXSDDateTimeStamp ?end.
                FILTER (?begin = '2020-01-01T00:00:00'^^xsd:dateTimeStamp)
        }
}
```

C.22

Endpoint: http://localhost:7200/repositories/[REPOSITORY]

```
PREFIX bag2: <https://bag2.basisregistraties.overheid.nl/bag/def/>
PREFIX bot: <https://w3c-lbd-cg.github.io/bot/#>
PREFIX bop: <https://alexdonkers.github.io/bop/index.html#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX seas: <https://w3id.org/seas/EvaluationOntology#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX neo: <https://sanderdemeij.github.io/neo/#>
select distinct ?building ?postcode ?energyLabel ?procedureName ?bouwjaar
?oppervlakte where {
        ?building bag2:gebruiksdoel ?gebruiksdoel.
        FILTER(?gebruiksdoel = 'woonfunctie' || ?gebruiksdoel = 'nan')
        ?building bag2:postcode ?postcode.
        ?building bag2:bouwjaar ?bouwjaar.
        ?building bag2:oppervlakte ?oppervlakte.
        OPTIONAL {
                ?building bop:hasProperty ?energyLabelProperty.
                ?energyLabelProperty bop:hasValue ?energyLabel.
                BIND('Actual' as ?procedureName).
        }
        OPTIONAL {
                ?building bop:hasProperty ?predEL.
                ?predEL bop:hasExecution ?execution.
                ?execution bop:usesProcedure ?procedure.
                ?procedure skos:prefLabel ?procedureName.
                ?execution bop:hasResult ?result.
                ?result bop:hasValue ?energyLabel.
        }
}
```