UNIVERSITY OF TARTU

Institute of Computer Science

Data Science Curriculum

Kristo Hõrrak

Environmental data asset and report generation management system

Master's Thesis (15 ECTS)

Supervisor(s): Urmas Hõrrak, PhD

Priit Adler, PhD

Tartu 2023

Environmental data asset and report generation management system

Abstract:

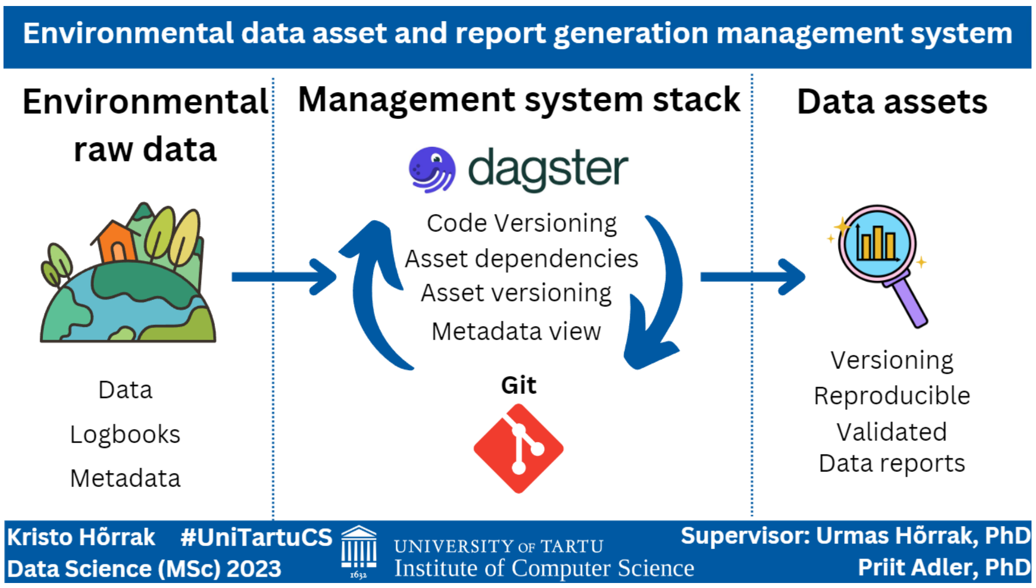
In this thesis, the primary objective was to implement a data management solution for the generation of time-series environmental data assets and reports. Overall, the system should aid in preparing for a more open-science approach to asset generation by allowing versioning, metadata annotation, and the ability to trace how and why data assets were created. In order to facilitate collaboration on such data-driven work, a technology stack consisting of multiple open source solutions were chosen. The solution should enable researchers to perform a variety of duties, including data collection, metadata injection, data versioning, and asset materialization, with relative ease. The implemented system makes it apparent how data assets are dependent on one another and how, by utilizing the tools, users can determine what type of data they are working with. As a result of implementing such a solution, an environmental data asset and report system was developed for two Tahkuse environmental measurement station gas analyzers, allowing for the generation of multiple data assets and reports in a clear, comprehensible, validated, versioned, and visually understandable way.

Keywords:

Data pipelines, Data management, Metadata, Data stewardship, Data versioning, Data assets, Dagster, Git, Data reporting, Python

CERCS: P170 Computer science, numerical analysis, systems, control

Visual abstract:



Keskkonnaandmete ja aruannete koostamise haldussüsteem

Lühikokkuvõte:

Antud magistritöö eesmärk oli rakendada avatud lähtekoodiga tarkvaralahendusi andmehalduslahenduse loomiseks keskkonnaandmete ja -aruannete genereerimiseks. Lahenduse põhieesmärk oli olla abiks avatud teaduse põhisele lähenemisele andmeobjektide ja aruandluse loomisel. Sellejaoks peab lahendus võimaldada genereerida andme objekte versioniseeritud viisil, pakkuda võimalust andmete metaandmetega rikastamiseks ja võimalust aru saada, kuidas andmete objektid üksteisest sõltuvad. Antud andmehalduse viis peaks teadlastel võimaldama täita mitmesuguseid ülesandeid, sealhulgas andmete kogumist, metaandmete sisestamist, andmete versioniseerimist ja objektide loomist. Antud lahenduse kasutamine peab aitama kasutajatel lihtsama vaevaga aru saada, milliste andmetega nad töötavad ja kuidas andmete formeerumine toimub. Töö juurutamise tulemusena töötati välja kahe Tahkuse seirejaama gaasianalüsaatori ja jaama enda jaoks keskkonnaandmete aegridade andmeobjektide ja aruandluse lahedus, mis võimaldab luua andmeobjekte selges, valideeritud, versioniseeritud ja visuaalselt arusaadaval viisil.

Võtmesõnad:

Andmehaldus, metaandmed, andmete versioniseerimine, andmearuandlus, Git, Python (Programmeerimiskeel), Dagster

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Visuaalne kokkuvõte:

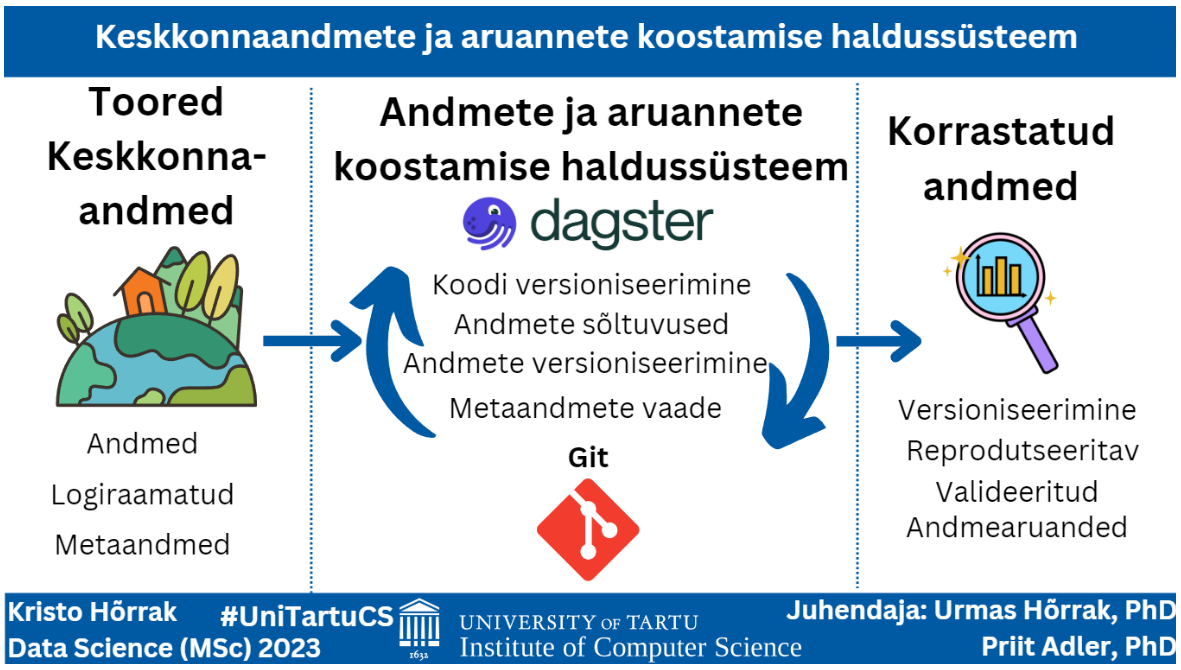


Table of Contents

Introduction

For a data-driven workflow to be well-organized, the system must be scalable, readable by others, and include version control and data testing. This is necessary for a stable system. Unfortunately, this is not the case for most data-related projects (Brown, Kathryn A. Kaiser , & David B. Allison, 2018). This is due to time constraints, writing code for a specific topic, not having to release the code for various reasons, etc. This leads to issues with the reproducibility of the completed project (Baker, 2016). To promote sharing of information and improved collaboration, the European Union has begun a program that promotes a more open approach to science (Union, 2023). The primary objective is to make the entire process more transparent by making all aspects of the research public, including the initial datasets, methodology, and source code, as well as the final datasets. Completely shifting from just writing an article to open-research managed science data flow would require a great deal of effort. This work attempts to show one of the numerous ways in which a transition to an open data workflow can occur. The primary objective is to demonstrate how this can be incorporated into environmental time series dataset workflows.

For this to take place, this work handles the following issues such as:

1. Provide a clear overview of how the data pipeline operates from beginning to end.
2. Easily update the dataset if the data or code has been changed. Inform the user if their dataset is not up to date to the current version.
3. Ability for the user to easily setup their own environment for a specific project and validate that the end results can be generated from the input (original) data.

This work implements a environmental time-series related workflow that take these tasks into design for generating data assets and data reports using the Dagster data orchestration platform (Dagster, Dagster main page, 2023) together with other technologies. At the end of the work, it should be clear how such an approach could make future implementations of comparable projects easier and less complicated to use. Using these techniques, one should be able to communicate their data workflow to others who are unfamiliar with the subject area with more simplicity.

1 Terms and Notations

Git VCS – Git Version Control system. A collection of tools for managing software development project version control.

Git VCS – Git Version Control system. A collection of tools for managing version control in software development projects.

Data asset – A data-related object.

Data Versioning - To identify what type of data is being worked with. Users can define it manually or by generating a checksum for the data asset.

Code Versioning – Defines the type of code version used for a given data asset. Typically added manually to define if an asset's code has changed significantly enough for the data version to differ from the previous version.

Data materialization – The production of a data asset.

Data propagation changes - Used to launch jobs related to data assets that require re materialization due to data, code or dependency changes.

Data pipeline - Sequence of data transformations from one format to another.

Metadata – Data that defines data. Additional data is typically utilized to assist comprehend what the data is about.

Technology stack – Stack of technologies used to build a system or application.

TIMESTAMPTZ – Timestamp with Time Zone information.

Pandas dataframe – Tabular formatted dataset used within the Python pandas data analysis library.

DOI – Digital unique identifier of an object

Dagit – Web UI for the dagster project

UI – User Interface

2 Proposed system requirements

In the initial stages of developing the proposed system for the generation environmental data assets and reports, a set of requirements were defined to make sure that the outcome of the project would result in a solution with future-proof value. These requirements were thoughtfully developed and evaluated to address key problems and challenges that arise when dealing with various kinds of time-series data. The main objectives of the defined requirements are to ensure that a wider range of users can adopt the solution and to streamline the data processing and analysis process in a research environment.

The following requirements were established to achieve the primary objectives:

1. The solution should incorporate a user-friendly interface for optimal usability, allowing users to navigate and comprehend the data pipeline with minimal effort. This means that by just looking at the pipeline, it is possible to figure out how the data is processed, when the data was processed, whether or not the data is up-to-date, and what the data assets dependencies are on one another.
2. There must be a guarantee that the processed data conforms to the necessary data validation requirements, thereby minimizing errors and inconsistencies between datasets and reports.
3. Testing capabilities must be present to validate the output of the pipeline and ensure that it complies with the intended output of the transformed data. This should also make it simple to modify the locations where files are loaded and unloaded, regardless of whether the files contain production or test data.
4. The proposed solution must make it possible for members of the team to work together, with the ability to exchange and alter data as well as code and processes.
5. There must be a guarantee that the pipeline's output is reproducible, meaning that it is possible to replicate the pipeline and obtain identical results between users if the initial data is the same. The solution should include versioning functionality, enabling users to monitor and manage changes to the pipeline and code. This should also enable users to revert to older versions and regenerate old datasets and reports based on outdated code, if necessary.
6. The solution should include a graphical user interface that enables users to plan the data pipeline in advance before interacting with actual data. This should enable users to plan for the generation of data assets, their inputs and outputs, and the metadata that is included with them.

7. It must be straightforward to install on a user's computer, with explicit instructions and minimal configuration required.

To meet these requirements, the proposed system should use open-source software solutions for code management, collaboration, and data orchestration. This should ensure that the project adheres to the latest norms, and if there are new versions of the tooling with beneficial new features, they can be incorporated into the solution whenever needed. This also enables the use of the same tools for other similar projects, as opposed to working on a custom project with a specific purpose.

3 System solution overview

The current iteration of the system for generating environmental data assets and reports incorporates the following technology stack to form an integrated solution: Python, Dagster, Papermill, Jupyter Notebooks, JupyterLab, Git, and an online git repository. Dagster is the primary component of the system, with the other components serving to manage and fulfill requirements for the system to function. The system consists of three primary components that are essential for its operation (Figure 1).

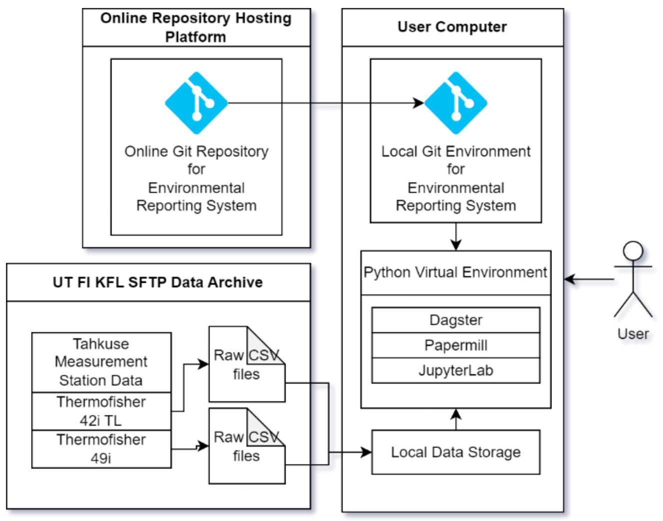

Online Repository Hosting Platform
Online Git Repository for Environmental Reporting System
User Computer
Local Git Environment for Environmental Reporting System
Python Virtual Environment
Dagster
Papermill
JupyterLab
User
UT FI KFL SFTP Data Archive
Tahkuse Measurement Station Data
Thermofisher 42i TL
Thermofisher 49i
Raw CSV files
Raw CSV files
Local Data Storage

Figure 1. Overview of the system's design.

First of all, we have the Online Repository Hosting Platform. This component contains the online Git repository, which contains the code and instructions required to get the local user environment running. This repository is accessible from any location with an internet connection, enabling collaboration and permitting users to work on the reporting system from multiple devices and locations. In this instance, GitHub was chosen as the online Git repository that enables users to maintain the code repository, handle version control, carry out code reviews, and evaluate issues with the reporting system. To prevent unauthorised users from gaining access to private code revisions, the GitHub repository was made private. In case it becomes necessary to make the repository public in the future, a separate repository containing code versions targeted for usage by others can be created.

Then we have the external data repository. Currently, the used data is from the Tahkuse air monitoring station (58°31.47′N, 24°55.53′E), which is one of the environmental research stations of the Laboratory of Environmental Physics (Est KFL) at the Institute of Physics (FI), University of Tartu (UT). These data are measured and recorded by the Nitrogen Oxide (NOx) Analyzer Model 42i-

TL and Ozone (O3) Analyzer Model 49i (Thermo Fisher Scientific Inc.) and are securely accessible from data repository by using the Secure File Transfer Protocol (SFTP). The raw measurement data files used as input for this project's data pipelines, as well as for generating the final data assets and data reports, are obtained from an external data repository. Since users should be able to work offline and generate datasets and reports, the project requires that users manually load the data onto their computers.

Finally, we have to consider the user's computer. The proposed stack was designed to run on any modern user's computer which runs Python 3.10. This allows users to execute data pipelines and generate desired outputs on their own machines. The user must have sufficient computational power to load datasets into memory during data processing. Poor system requirements could cause the system to malfunction or operate inefficiently. The solution's software should be compatible with any Mac, Linux, or Windows environment that supports the required Python and Git versions. No additional software is required to operate the solution, making the installation less resource-intensive and requiring fewer steps to get it operating.

3.1 Installation of the system

The required workflow to get the current system installed and running is explained below.

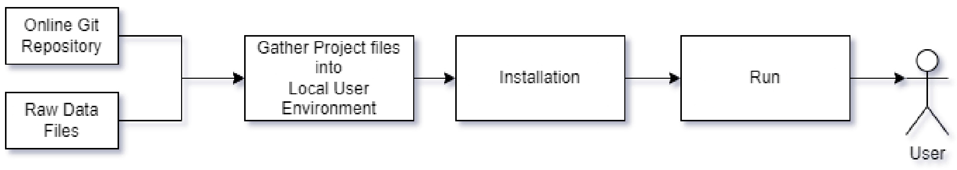


Figure 2. The workflow required to get the current system up and operating.

The PyCharm IDE is used for the installation (Figure 2) and configuration of this project. PyCharm IDE is used mainly due to its built-in Git tool and simple Python virtual environment configuration. Git support in PyCharm simplifies tracking changes within the code and provides a simpler understanding of who has changed what. Users can simply commit modifications to a local Git repository and push them to an online repository. PyCharm makes it easier to create virtual environments for specific Python versions. Users can install and maintain the data management solution dependencies and packages, which makes the system reliable and consistent over time. Using PyCharm, you can also easily access the virtual environment through the operating systems terminal.

The project currently runs on Python version 3.10 and was tested on Windows 11. The system should most likely also easily run on other systems due to the lack of operating system-specific requirements. After cloning the project files from GitHub, the following libraries must be installed in the virtual Python environment in order to run the project.:

- "pandas~=1.5.3",
- "numpy~=1.23.5",
- "dagster~=1.3.2",
- "duckdb~=0.7.1",
- "dagstermill~=0.19.2",
- "jupyterlab==3.6.3",
- "matplotlib==3.7.1",
- "seaborn==0.12.2",
- "calplot==0.1.7.5",
- "dagster_pandas==0.19.3"

The required libraries and their versions are also specified in the "setup.py" file within the "env-report-gen" folder of the project.

After the initial project cloning has been done, the system can be installed by navigating to the folder named ‘env-report-gen’ within the project and running the command **'pip install -e ".[dev]"**', which downloads the required project dependencies. This command is typically only required to be executed once. If the library dependencies changed, the command would need to be executed again in order to update the necessary libraries. After that, running the command **'dagster dev’** in the terminal should launch a local web interface accessible through **http://localhost:3000**, which can be used to view the project.

3.2 Initial raw data

The initial data, which was used for this project, was collected at the Tahkuse Measurement Station by the Environmental Physics research group of the Institute of Physics at the University of Tartu. The initial datasets contain raw measurement files for the entire year of 2022 from the ThermoScientific Model 49i Ozone (O3) Analyzer and ThermoScientific Model 42i (NO-NO2-NOx) Analyzer. There are also logbook files for both the devices and the station in addition to the gas analysers’ data. Initial project data are organised within folders as follows:

./report_dataset/in/{station_name}/{device|station}/{device_name}/{logbook|raw_data}/ .

The levels in file path/route represent the following:

- report_dataset - Subfolder within the project to separate the report dataset from other files.

- in - Defines that the data should be used as input
- {station_name} - defines from which station the data originates from
- {device|station} - defines if the data is linked to the station itself or a measurement device.
- {device_name} - name of the device, in this case thermofisher_49i or thermofisher_42iTL
- {logbook|raw_data|dataset} - defines what kind of data is being stored. If it is related to logbook entries or raw_data or newly generated datasets.

3.3 Choice of technologies and libraries

3.3.1 Git

Git is a reliable version control system (VCS) that allows developers to keep track of the development of their source code (Git, 2023). This allows users to readily identify who made changes to a file and when. This is especially essential when working in a team, as it enables collaboration and ensures that everyone is working on the same code version. Git's primary function in the current work is to track data assets along with asset code and data versioning functionality. By being able to manage this, users can easily rollback to previous versions of any data asset in order to recreate the asset as it existed at the time. This makes it simple for users to collaborate and make necessary adjustments. Using git, users can also work on multiple branches, allowing them to merge only the changes that are deemed beneficial for them. For example, a developer might want to implement new logic that changes the data asset totally, which might not be beneficial for a user who wishes to complete his work with the current data asset.

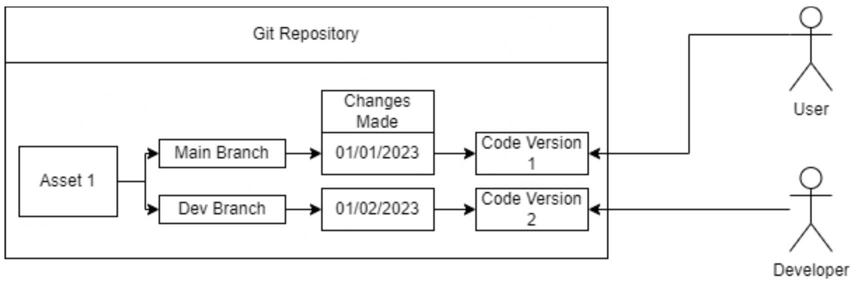


Figure 3. Example of a user and developer working on different data assets based on code versioning.

3.3.2 Dagster

Dagster has been selected as the data orchestrator for this project. This is due to the platform's multiple different built-in integrations and due to the fact that it can be easily installed to any users computer with basically only three commands if they have the required software preinstalled (Dagster, Create new project, 2023).

3.3.2.1 Data pipeline visualisation

Dagster allows us to visualise our data pipeline based on code. This allows to easily identify which data assets are generated and their dependencies on eachother. This also helps the preparation of the data pipeline by planning out the required data assets and their dependencies.

3.3.2.2 Separation of file resources and I/O from data assets

Dagster's resources and I/O managers allow us to separate our data assets from data files. We don't have to worry about how our data is stored and loaded when we use this. This also allows us to separate the file logic and rewrite it based on the use case. We can reconfigure and rewrite the resources and I/O managers based on the needs of those using it. If we want to store the data in a different folder or in a completely different way, we can do so without interfering with the data pipeline code itself.

3.3.2.3 Asset versioning and dependencies

Dagster includes dependency handling for code, data versioning, and asset dependencies. If an asset code version, data version, or dependencies change, the platform notifies the user. This enables us to propagate changes down the chain for all other assets that rely on this specific asset and must be kept up to date. Together with Git, this gives us a way to ensure that our data is always in sync with the git repository. Dagster will notify us if our dependencies differ from the currently materialized assets when we fetch, update, or change branches within the git repository and the code versioning for assets were to change.

3.3.2.4 Asset metadata, description, and metadata runtime graphs

It is possible to add metadata and descriptions to our data assets using Dagster. This aids in providing additional external information about the asset. In addition, it is possible to generate metadata about the asset and provide additional information in the form of metadata graphs that Dagster's user interface can display. Additionally, Dagster provides some of its own metadata by default when materialising assets, such as data version and the utilized data asset input versions, to provide insight into whether an assets data has been changed. Additionally, such behaviour can be rewritten with custom logic if desired.

3.3.2.5 Asset data validation

In addition, Dagster allows users to set up data validation for their data assets. This feature is shown together with metadata and description about the asset in the user interface. By making use of validation, it is possible to establish rules that run after an asset has finished its work to materialize and verify that it is in the required format. The asset will not materialize and will fail if it does not meet the data validation criteria.

3.3.3 Jupyterlab & Jupyter Notebooks

JupyterLab is an open-source platform for working with Jupyter notebooks and the Python programming language. It allows users to load datasets and code, look into, visualize, and comment on their findings using markdown and code cells (Jupyter, 2023). Each code cell's outputs can be saved and displayed, allowing users to share their findings without requiring other users to execute the notebooks to view the results. Jupyterlab was chosen for this system because it enables users to interact with Dagster-generated datasets. This also eliminates the possibility that incompatible library package versions between the Dagster environment and the user's environment will eventually cause problems. Using Jupyter notebooks, we can reuse users' notebooks as templates for the data reports that Dagster can use via the Dagstermill library/package to generate data reports for the reporting system.

3.3.4 Papermill/Dagstermill

Papermill is a tool for parameterizing and executing Jupyter Notebooks (nteract, 2023). For the current solution, we use a Dagster-specific version named Dagstermill, which has internal integrations such as the ability to view notebook templates and run results from within the Dagster user interface. Dagster injects additional code within the notebook, allowing users to interact with the data assets from within the notebook itself, enabling the parameterized notebooks to be executed and debugged as needed.

3.3.5 DuckDB

DuckDB is an embedded analytical database that is free and open source. DuckDB allows users to use SQL syntax to query local CSV, parquet, and JSON files in bulk and return the results in Pandas dataframe format (Dennis, 2023). DuckDB was used as an internal resource in this project to gather data from logbook files and clean the data of gas analyzers. This tool made it very easy to collect the necessary information in the correct format for the data pipelines.

3.3.6 Graphing libraries

Seaborn, Calplot, and Matplotlib are used as graphing libraries for this project. Calplot allows users to generate a calendar-type view of their data. For this project, notebook templates use Calplot to display how much data was acquired for that time period. Matplotlib allows users to generate plots from a variety of different kinds of data. Matplotlib is also integrated into Pandas using the plot function, allowing for easy visualization of data. Seaborn is a library that's built on top of Matplotlib and provides extra functionality for generating more different types of plots compared to Matplotlib.

4 Usage of the system for environmental data assets and data reports.

The final system consists of the local data repository and the Dagster project managed by Git version control. Utilizing the configured data processing pipeline, the system is able to make use of the local raw data files by reading and writing data using Dagster's resources. The data pipeline consists of data assets that employ Dagster features for metadata, metadata graphs, data validation, versioning, and asset dependencies. The system generates data assets in an organized manner and allows users to generate data reports from the current data assets using the predefined notebook template files.

4.1 Input data format overview

The input data used for this system consists of three different formats. The measurement devices (gas analyzers) and station logbooks follow the same JSON format, and both the measurement devices have their own CSV files with different columnar features (gas concentrations, meteorological and technical parameters) for the raw data. The data are measured and recorded with 10 second timestep.

4.1.1 Logbooks

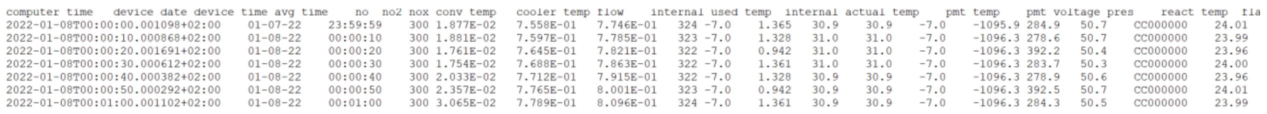
The logbook folders contain JSON files with entries about events that have happened with devices or at the station (notes about the service and calibration of devices, observations on the environment and meteorological events, pollution episodes, etc.). These events are in JSON record format and contain the following information:

- start_time – String type data about event start in TIMESTAMPTZ format.
- end_time - String type data about event end in TIMESTAMPTZ format
- entry_by – String type data about the name of the person who made the entry.
- type – String type categorical data about what event took place.
- action - String type data about what action took place for the event.
- description- String type data about what took place.
- authors –JSON format freestyle data about what the event authors wish to share.

4.1.2 CSV Files

The raw data files are in daily partition CSV file format with a 10-second timestep. A specific measurement collection software, which communicates with the measurement device to gather data from the device itself, collects the raw data files. The raw CSV files contain some features that are similar for the files of different gas analyzers and some that are not due to having different working parameters to record.

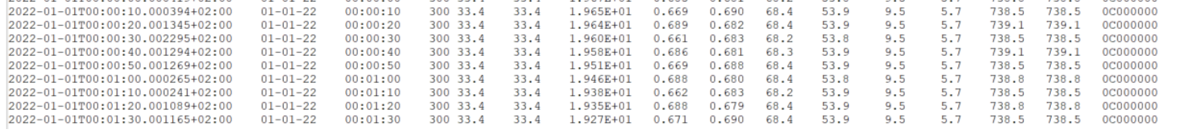
The raw data files (Figure 4) for the ThermoScientific Model 42i contain 47 different recorded variables. These include TIMESTAMPTZ, float, categorical features, hex flags, and strings, among others. The most important is "computer time" which contains the measurement's time in TIMESTAMPTZ format. Then we have the device's primary outputs in float format, such as "no", "no2", and "nox", which provide the gas concentration in the units specified in the string categorical column "gas unit." The majority of the remaining units are of diagnostic variables, such as "operation mode", "flags" which can be used to determine if the device is functioning properly. Typically, when these variables are changed during device maintenance by the device itself, there is also an entry in the logbook that provides user-specific information about what occurred in finer detail.



```
computer time   device date device time avg time    no  no2 nox conv temp   cooler temp flow    internal used temp  internal actual temp    pmt temp    pmt voltage pres    react temp  fla
2022-01-08T00:00:00.001098+02:00    01-07-22    23:59:59    300 1.877E-02   7.558E-01   7.746E-01   324 -7.0    1.365   30.9    30.9    -7.0    -1095.9 284.9   50.7    CC000000    24.01
2022-01-08T00:00:10.000868+02:00    01-08-22    00:00:10    300 1.881E-02   7.597E-01   7.785E-01   323 -7.0    1.328   31.0    31.0    -7.0    -1096.3 278.6   50.7    CC000000    23.99
2022-01-08T00:00:20.001691+02:00    01-08-22    00:00:20    300 1.761E-02   7.645E-01   7.821E-01   322 -7.0    0.942   31.0    31.0    -7.0    -1096.3 392.2   50.4    CC000000    23.96
2022-01-08T00:00:30.000612+02:00    01-08-22    00:00:30    300 1.754E-02   7.688E-01   7.863E-01   322 -7.0    1.361   31.0    31.0    -7.0    -1096.3 283.7   50.3    CC000000    24.00
2022-01-08T00:00:40.000382+02:00    01-08-22    00:00:40    300 2.033E-02   7.712E-01   7.915E-01   322 -7.0    1.328   30.9    30.9    -7.0    -1096.3 278.9   50.6    CC000000    23.96
2022-01-08T00:00:50.000292+02:00    01-08-22    00:00:50    300 2.357E-02   7.765E-01   8.001E-01   323 -7.0    0.942   30.9    30.9    -7.0    -1096.3 392.5   50.7    CC000000    24.01
2022-01-08T00:01:00.001102+02:00    01-08-22    00:01:00    300 3.065E-02   7.789E-01   8.096E-01   324 -7.0    1.361   30.9    30.9    -7.0    -1096.3 284.3   50.5    CC000000    23.99
```

Figure 4. Snippet from ThermoScientific Model 42i-TL NOX Analyzer raw data file.

The ThermoScientific Model 49i O_3 Analyzer raw data files (Figure 5) contain 41 distinct variables. Some of the variables have the same format as the ThermoScientific Model 42i-TL NOx Analyzer because they are collected by the same kind of data acquisition software, but there are differences in the collected files because the device uses different measurement method/principle and working parameters. The "computer time" and some diagnostical features such as "mode" and "flags" are similar, but the measurement units are distinct due to the measurement of different gases.



```
computer time   device date device time avg time    bench used temp bench actual temp   o3  flow a  flow b  o3 lamp temp    bench lamp temp lamp voltage bench  lamp vo
2022-01-01T00:00:00.000419+02:00    01-01-22    00:00:00    300 33.4    33.4    1.967E+01   0.685   0.681   68.2    53.8    9.5     5.7     738.8   738.8   0C000000
2022-01-01T00:00:10.000394+02:00    01-01-22    00:00:10    300 33.4    33.4    1.965E+01   0.669   0.690   68.4    53.9    9.5     5.7     738.5   738.5   0C000000
2022-01-01T00:00:20.001345+02:00    01-01-22    00:00:20    300 33.4    33.4    1.964E+01   0.689   0.682   68.4    53.9    9.5     5.7     739.1   739.1   0C000000
2022-01-01T00:00:30.002295+02:00    01-01-22    00:00:30    300 33.4    33.4    1.960E+01   0.661   0.683   68.2    53.8    9.5     5.7     738.5   738.5   0C000000
2022-01-01T00:00:40.001294+02:00    01-01-22    00:00:40    300 33.4    33.4    1.958E+01   0.686   0.681   68.3    53.9    9.5     5.7     739.1   739.1   0C000000
2022-01-01T00:00:50.001269+02:00    01-01-22    00:00:50    300 33.4    33.4    1.951E+01   0.669   0.688   68.4    53.9    9.5     5.7     738.5   738.5   0C000000
2022-01-01T00:01:00.000265+02:00    01-01-22    00:01:00    300 33.4    33.4    1.946E+01   0.688   0.680   68.4    53.8    9.5     5.7     738.8   738.8   0C000000
2022-01-01T00:01:10.000241+02:00    01-01-22    00:01:10    300 33.4    33.4    1.938E+01   0.662   0.683   68.2    53.9    9.5     5.7     738.5   738.5   0C000000
2022-01-01T00:01:20.001089+02:00    01-01-22    00:01:20    300 33.4    33.4    1.935E+01   0.688   0.679   68.4    53.9    9.5     5.7     738.8   738.8   0C000000
2022-01-01T00:01:30.001165+02:00    01-01-22    00:01:30    300 33.4    33.4    1.927E+01   0.671   0.690   68.4    53.9    9.5     5.7     738.5   738.5   0C000000
```

Figure 5. Example snippet from the ThermoScientific Model 49i Ozone Analyzer raw data file.

4.2 Data pipeline overview

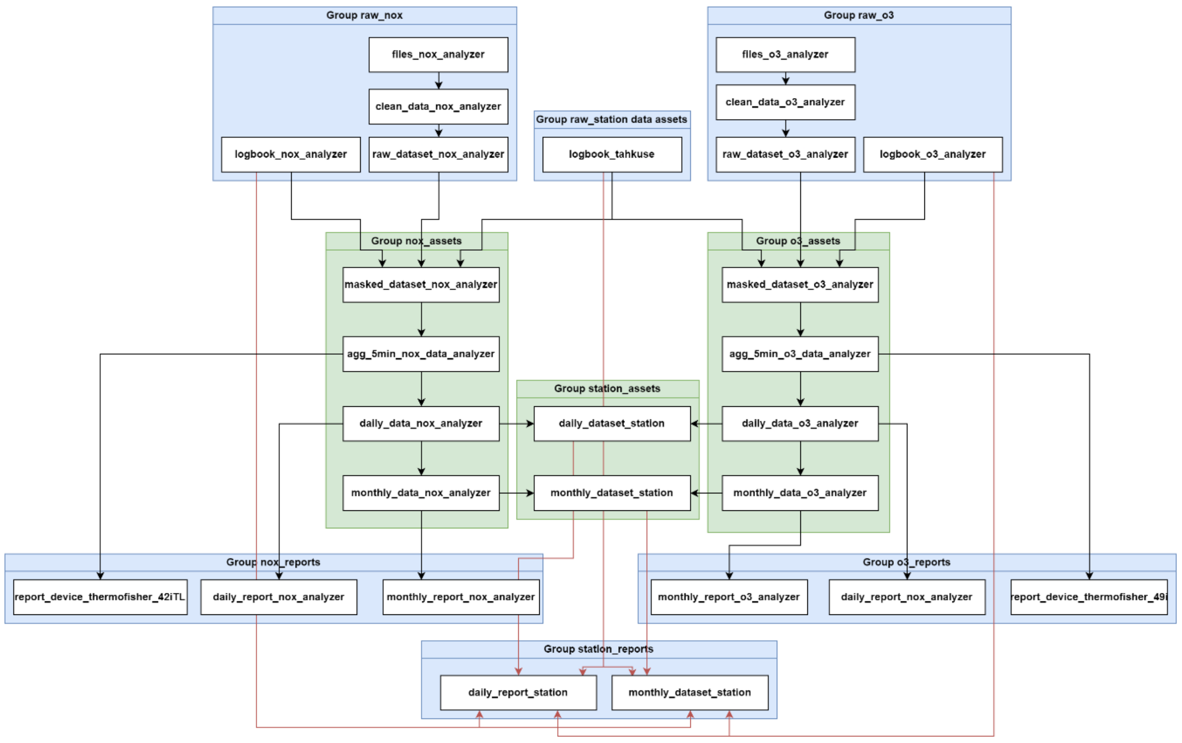


Figure 6. Overview of the entire data pipeline for the Tahkuse measurement station data assets and data reports. The arrows represent asset dependencies on one another.

In general, the data pipeline consists of nine asset groupings (Figure 6) which are used to generate 27 data assets which contain 8 data reports. The assets are categorized in accordance with their device or station type and asset category. The suffix "raw" separates raw data processing that requires more computational resources into separate groups. Due to having been archived beforehand, raw data should not typically require rematerialization that frequently. The suffix "assets" identifies data assets that use the Metadata I/O manager to store files as data assets according to the metadata configuration. The suffix "reports" defines data reports that visualize data variables using the previously created assets from the "assets" group and notebook template files. The lines on Figure 6 represent how the asset dependencies are defined.

4.2.1 Resources & I/O Managers

To separate data asset logic from I/O operations, custom resources are developed in the present work for accessing the raw data files and writing data assets together with data reports to the desired locations.

4.2.1.1 LocalEventResource

This resource handles the access to the logbook information. For configuration parameters it's possible to set the data input folder where the data repository raw data files reside and the timezone in which the returned Pandas dataframe should be in. This resource takes in the following variables:

- Location – Used to query data from a specific measurement station.
- Group – Used to identify if we are dealing with a measurement station or a measurement device.
- Name – Used to identify if we are dealing with measurement station events logbook or a measurement device logbook.
- Type - Identifies what folder name is used as the logbook.

Based on these variables, we are able to generate the path to the specified logbook directory and query its contents. Using DuckDB, the original JSON format logbook files are converted into a single Pandas dataframe with a predefined data format, as JSON files are not reliable by default. This is due to the fact that users can write any type of data into them. Using DuckDB's functionality to set the timezone, we can convert the original timestamps to the format specified in the configuration: "***SET TimeZone='self.timezone'***;". After that we can query from all the JSON logbook files the information we require as the specified data type within the resource function.

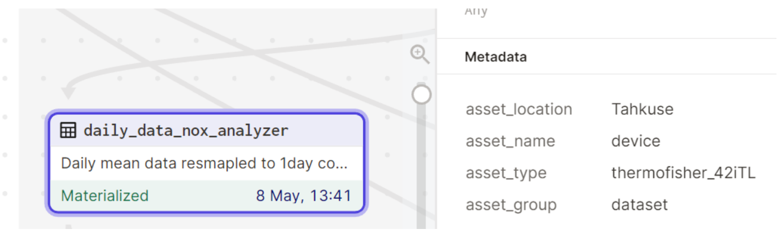
4.2.1.2 LocalDeviceFileScan

This resource is used to manage the raw data files generated by the gas analysers. The resource is configured with a path that specifies the location of the data assets repository. The resource accepts the device's location and name and returns a Pandas dataframe containing the file paths and file sizes of every file in the specific directory.

4.2.1.2 LocalMetadataParquetManager

This is the primary I/O manager used in this project for storing Pandas dataframe-type data assets. The resource requires that each data asset have predefined metadata describing its location, name, type, and group. Using this information and the predefined configuration of where the data asset output should be the storage path for the data assets is defined.

For instance, the ***daily_data_nox_analyzer*** data asset is stored in the path *"report_dataset/out/Tahkuse/device/thermofisher_42iTL/dataset/daily_data_nox_analyzer.parquet"* in the project directory. This is because the primary resource configuration path is *"../report_dataset/out"* and the asset metadata is defined as shown in Figure 7.



daily_data_nox_analyzer

Daily mean data resmapled to 1day co...

Materialized 8 May, 13:41

Metadata

asset_location	Tahkuse
asset_name	device
asset_type	thermofisher_42iTL
asset_group	dataset

Figure 7. Data asset which uses LocalMetaDataParquetManager and its metadata. Image taken from within the Dagit UI.

4.2.2 Assets

Since most of the assets are designed to perform a similar function but contain distinct logic due to internal data differences, they are described together and separated using curly bracket symbols “{“, “}”and vertical bar symbols “|”. So, if asset_{A|B} exists, the actual data pipeline contains both assets: asset_A and asset_B.

4.2.2.1 logbook_{nox_analyzer|o3_analyzer|tahkuse}

These assets use the LocalEventResource to retrieve information about the logbook for a specific device or station. Requires the data to be in the EventDataFrame format, which requires the start_time and end_time fields to be in datetime64[ns, Europe/Tallinn] format, the type of the field to be in categorical format ['warning', 'note', 'failure','maintenance'], and the action and description fields to be in string format. If any of these conditions are not met, the materialization of the asset will fail. The asset also returns metadata graphs about how many entries were found about the logbook.

4.2.2.2 files_{nox_analyzer|o3_analyzer}

These assets use the LocalDeviceFileScan resource to collect data on all raw data files related to the specified gas analyser. The assets return the same dataframe queried from the resource and validates the filepaths as strings and the filesizes as float data type. Additionally, the assets return metadata indicating how many files were found and how much space they occupy on the disk.

4.2.2.3 clean_data_{nox_analyzer|o3_analyzer}

Both assets depend on the gas analyzer specific **files_{nox_analyzer|o3_analyzer}** assets to load, clean, and normalize the data files before saving them as parquet files in the output folder. As metadata, the number of converted files, file size in parquet format, and compressed space is saved and displayed on metadata graphs. As with **files_{nox_analyzer|o3_analyzer}** assets, the validation requires the new file paths and file sizes to be validated as string and float data types.

4.2.2.4 raw_dataset_{nox_analyzer|o3_analyzer}

These assets gather all the data from **clean_data_{nox_analyzer|o3_analyzer}** separate parquet files into one single file. The assets contain fewer variables than before due the lack of need to utilize all the fields. The data validation schema shows/determines what variables are currently available in the asset. As metadata, the Pandas dataframe memory usage is given, along with the data how many rows and columns the Pandas dataframe contains.

4.2.2.5 masked_dataset_{nox_analyzer|o3_analyzer}

The asset combine the gas analyzer logbook, station logbook together with the raw dataset to mask out events that have happened turning the **raw_dataset_{nox_analyzer|o3_analyzer}** recording/measurement time. In addition, both assets also have automatic bad data masking logic on the dataset to display any bad data actor timeseries. These procedures are based on extra diagnostic variables of devices. In the data validation, in addition to original data fields the extra mask columns based on the station logbooks, measurement devices and automatic filtering are added. The primary sum of all the invalid rows is displayed in the mask **"bad_data"** integer column.

4.2.2.6 agg_5min_{nox_analyzer|o3_analyzer}

This data asset uses the masked data asset to filter out all the rows that could contain the bad quality data and related information. In addition, unnecessary fields required for generating the reports and other assets are also removed to reduce redundant information within the datasets. Also, the dataset is resampled from the raw 10-second timestep to 5-minute timestep. Aggregated values of min, max, mean, count, and standard deviation are returned. If it is found that an aggregated bucket count contains fewer readings than 80%, it is thrown out of the dataset.

4.2.2.7 report_device_{thermofisher_42iTL|thermofisher_49i}

These assets are used to generate a notebook report for the **agg_5min_{nox_analyzer|o3_analyzer}** data assets. Both of the assets have their own predefined notebook templates. The notebooks contain instructions for loading the data analysis and graphing libraries and cell tags to define where Papermill may load the data assets data. The notebooks contain templating information to inform the user of what device the data is visualized from, what the dataset consists of basic information to describe the columns within the dataset, and dataset memory usage. For the graphing part, most of the important variables are plotted using the min, max, mean values. The validated amount of data per time bucket is displayed using a calendar plot. The relationship between (selected) variables is found out and the correlation information is also given using a Pearson correlation plot and a heatmap.

4.2.2.8 daily_data_{nox_analyzer|o3_analyzer}

These assets depend on the **agg_5min_{nox_analyzer|o3_analyzer}** asset to generate a daily timestep summary of the main gas analyzer measurement parameters without any diagnostical/technical data. The days that contain less than 80% data from the total time period are considered as invalid and removed from the dataset. The metadata shows how many of the resampled fields don’t contain enough information to be validated. The metadata also shows how much memory the data asset takes to be loaded into memory.

4.2.2.9 daily_report_{nox_analyzer|o3_analyzer}

These assets are used to generate a notebook report for **the daily_data_{nox_analyzer|o3_analyzer}** data assets. As with the previous notebook materialization assets, this report consists of statistics displaying how much data there is per time bucket and the min, mean and max values and corresponding graphs of measurement variables.

4.2.2.10 monthly_data_{nox_analyzer|o3_analyzer}

The asset contains resampled monthly data from the gas analyzers datasets to utilize the **daily_data_{nox_analyzer|o3_analyzer}** to gather monthly min, max, mean, and count values about the gases. It is required for a month to have more than 70% of the readings to be valid.

4.2.2.11 monthly_report_{nox_analyzer|o3_analyzer}

The monthly report visualizes the data count and gas measurement values from the monthly data asset.

4.2.2.12 {daily|monthly}_dataset_station

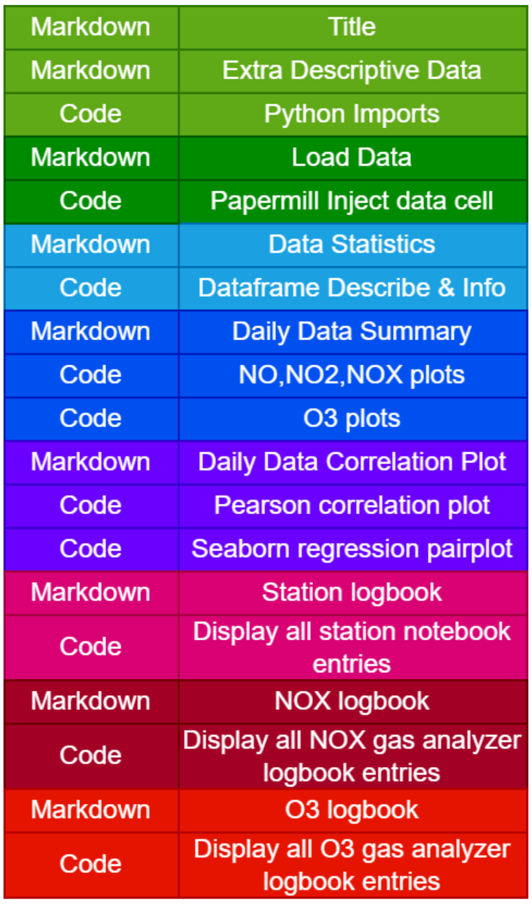
This dataset is a joined dataset from both gas analyzers daily or monthly dataset assets. The assets are joined by the time index.

4.2.2.14 {daily|monthly}_report_station

The daily and monthly station report utilizes all the available logbook and the **daily_dataset_station** data assets to visualize both the dataset and the available information in the logbooks. In addition extra text is given in the beginning about the measurement arrangement, where and how the measurements were carried out and what kind of devices are used.

4.3 Templates overview

Each data asset report has been provided with its own Jupyter notebook template file. This came from the need to handle specific changes more easily within the assets and to manage the version control and visualize various assets together with their corresponding functions. All of the reports follow the same kind of structure. Each notebook must contain markdown information describing the type of code that will be executed in the following cells. Before importing the actual data, each notebook must also have the library imports at the top. Figure 8 is an illustration of a prototype structure for the Daily Station notebook template, in which each code cell contains a markdown cell to notify the user of the type of information that will be displayed.



Markdown	Title
Markdown	Extra Descriptive Data
Code	Python Imports
Markdown	Load Data
Code	Papermill Inject data cell
Markdown	Data Statistics
Code	Dataframe Describe & Info
Markdown	Daily Data Summary
Code	NO,NO2,NOX plots
Code	O3 plots
Markdown	Daily Data Correlation Plot
Code	Pearson correlation plot
Code	Seaborn regression pairplot
Markdown	Station logbook
Code	Display all station notebook entries
Markdown	NOX logbook
Code	Display all NOX gas analyzer logbook entries
Markdown	O3 logbook
Code	Display all O3 gas analyzer logbook entries

Figure 8. Template prototype of the daily station notebook for showing how the data structure should be for the report.

To use notebooks as a template, the initial data must be defined in a single cell with the notebook cell tag "parameters" to allow Dagstermill to inject data assets into the notebook. The notebook code should then execute normally in a sequential order and produce the outputs of the code cells, which in turn return the desired visualizations. Also, by utilizing Dagster, it is possible to view the most recent notebook materialization from within the Dagster Web User Interface Dagit.

5 Evaluation of the system

Overall, all requirements specified for the environmental data asset and report generation management system have been met and the goals have been achieved. As a result, we have a robust, scalable and version-controlled data asset generation system that simplifies the creation of data assets, as well as comprehension of how specific data assets and data reports are generated. This should make it simpler for a variety of users to understand how the data assets were made and what their intended purposes are.

5.1 Data visualization and prototyping

Using Dagster in the environmental data asset and report management allows for simpler way to visualize the dependencies between data assets (Figure 9). Since we don't have to deal with I/O management on the asset side of things, therefore we can easily start prototyping our data pipeline and the required data assets.

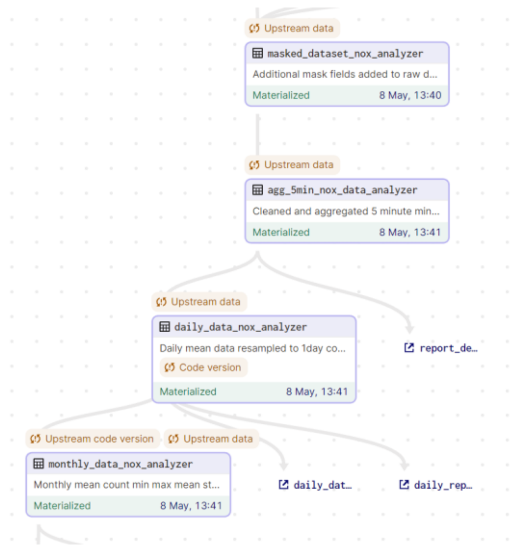
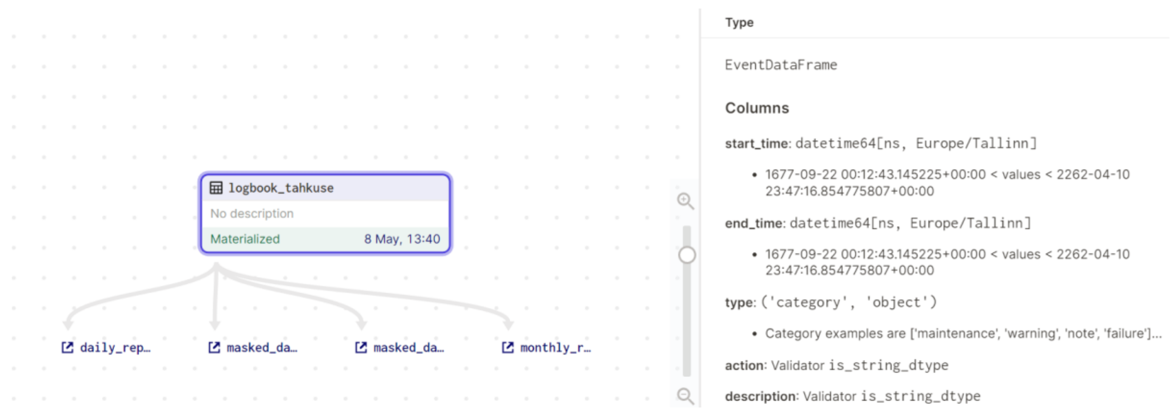


Figure 9. How the assets dependencies are visualized in Dagsters UI Dagit.

By designing the data pipeline in this manner, we can generate a test environment with sample data in advance to determine if our data assets are in the correct dependency chain and if their formats and metadata with descriptions are complete. Utilizing metadata graphs, we can also predict and calculate the extent of future data requirements.

5.2 Data Validation

By incorporating data validation into the data assets system, we can be certain that our data assets always satisfy the predetermined quality requirements. This, when combined with the user interface, allows us to clearly indicate to users what type of data the data asset contains (Figure 10). If something goes wrong, for example, if a user entered data in the incorrect format by failing to satisfy the categorical requirements, the system would not materialize the asset and all dependent assets. The system would inform us with a log trace about information about what went wrong within the data materialization process. This saves users time, ensures that the data is correctly materialized, and utilizes the most recent code and data versioning to provide users with security that the data assets they are utilizing are correct.



Type

EventDataFrame

Columns

start_time: datetime64[ns, Europe/Tallinn]

- 1677-09-22 00:12:43.145225+00:00 < values < 2262-04-10 23:47:16.854775807+00:00

end_time: datetime64[ns, Europe/Tallinn]

- 1677-09-22 00:12:43.145225+00:00 < values < 2262-04-10 23:47:16.854775807+00:00

type: ('category', 'object')

- Category examples are ['maintenance', 'warning', 'note', 'failure']...

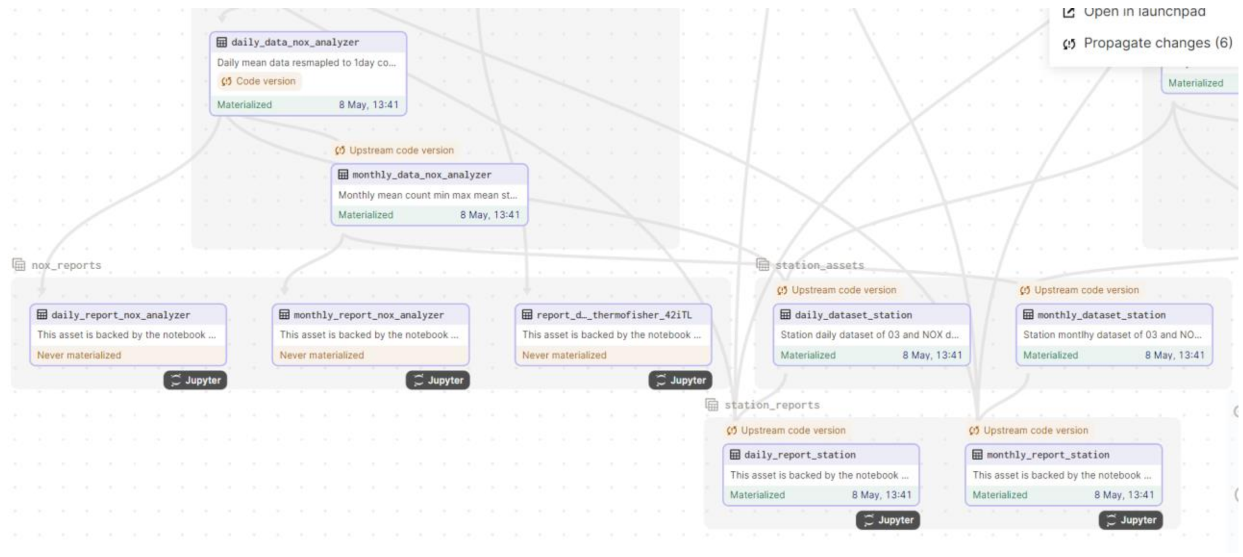
action: Validator is_string_dtype

description: Validator is_string_dtype

Figure 10. Tahkuse logbook asset using the predefined EventDataFrame data type to define what kind of columns with rules are valid within the asset. Image taken from Dagit UI.

5.3 Data, Code and Dependency Versioning.

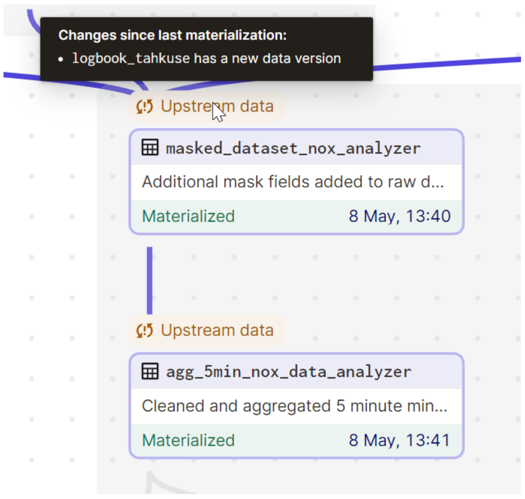
By utilizing data, code, and dependency versioning together with Git, we can always be certain that our assets are as expected. This, along with Dagster's ability to propagate changes, makes it straightforward to determine which assets need to be rematerialized without having to rerun the entire data pipeline (Figure 11).



Open in launchpad
Propagate changes (6)
daily_data_nox_analyzer
Daily mean data resmapled to 1day co...
Code version
Materialized 8 May, 13:41
Upstream code version
monthly_data_nox_analyzer
Monthly mean count min max mean st...
Materialized 8 May, 13:41
nox_reports
station_assets
daily_report_nox_analyzer
This asset is backed by the notebook ...
Never materialized
Jupyter
monthly_report_nox_analyzer
This asset is backed by the notebook ...
Never materialized
Jupyter
report_d..._thermofisher_42iTL
This asset is backed by the notebook ...
Never materialized
Jupyter
Upstream code version
daily_dataset_station
Station daily dataset of 03 and NOX d...
Materialized 8 May, 13:41
Upstream code version
monthly_dataset_station
Station montlhy dataset of 03 and NO...
Materialized 8 May, 13:41
station_reports
Upstream code version
daily_report_station
This asset is backed by the notebook ...
Materialized 8 May, 13:41
Jupyter
Upstream code version
monthly_report_station
This asset is backed by the notebook ...
Materialized 8 May, 13:41
Jupyter

Figure 11. Code versioning change displayed with propagate changes option. Image taken from Dagit UI.

If we are working in a collaborative environment and other users make changes, which we want to see, we can figure out which assets were modified without having to examine the source code (Figure 12). This is also beneficial if we add more system dependencies in the future.


Changes since last materialization:
- logbook_tahkuse has a new data version

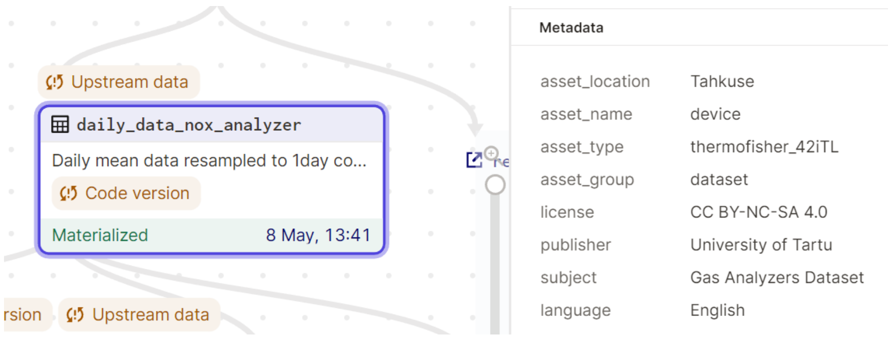
Additional mask fields added to raw d...

Cleaned and aggregated 5 minute min...

Figure 12. Dagit UI notification that the data versioning change due to new data loaded in.

5.4 Metadata

In addition to the utilized data assets, metadata was added to environmental data assets. (Figure 13). This allows other users to understand valuable information about the datasets, such as their licensing agreements, and in the future, these Metadata assets could be used to link our assets with a DOI-based data platform. With version control, we are able to make sure that the current Dagster-generated data versioning identifier corresponds to the metadata, and that our data is on the correct data version, along with useful metadata such as the description, run timestamp and code version.



Daily mean data resampled to 1day co...

Metadata

asset_location	Tahkuse
asset_name	device
asset_type	thermofisher_42iTL
asset_group	dataset
license	CC BY-NC-SA 4.0
publisher	University of Tartu
subject	Gas Analyzers Dataset
language	English

Figure 13. Metadata visualization within Dagit UI for the NOX analyzer daily dataset.

In addition, Dagster enables the visualization of metadata assets as graphs to provide users with information regarding the evolution of their assets. Figure 14 shows how the environmental data report system uses this in multiple assets to monitor the overall data asset volumes and metadata, such as row counts, so that users of the datasets can be notified if there have been changes over time.

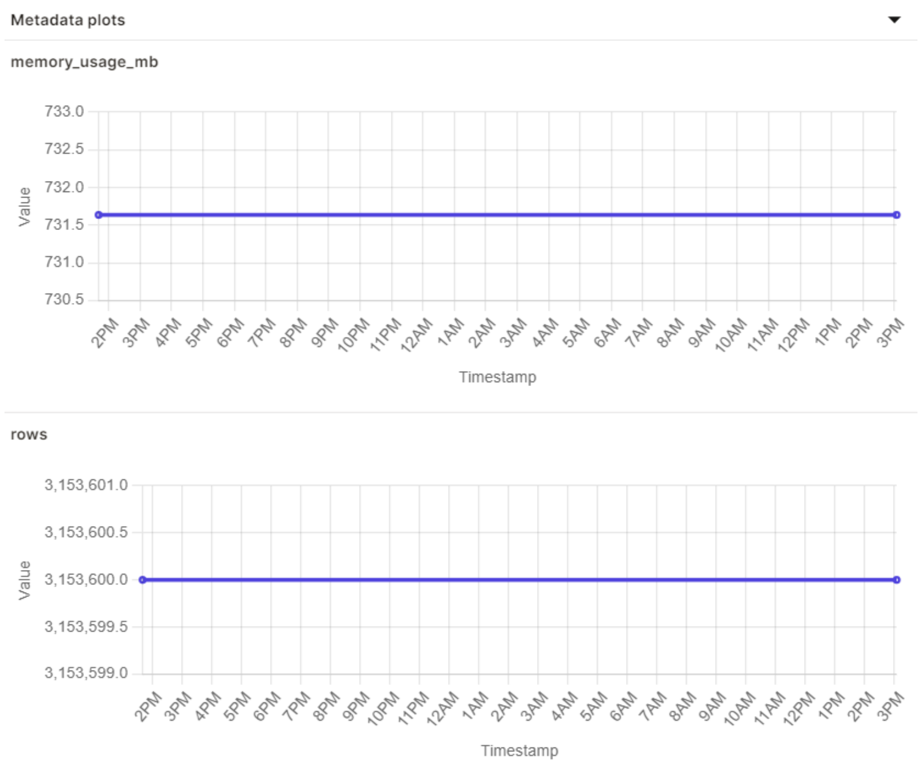


Figure 14. Metadata plots which have had no changes between materialization show within Dagit UI.

5.5 Data assets and notebook reports

The data reports generated through the templates were generally successful. Dagster has built-in support for browsing the source and materialized notebooks, which is a significant advantage for users who lack the software required to open Jupyter notebook ipynb-format files and quickly wish to view the materialized asset (Figure 15).

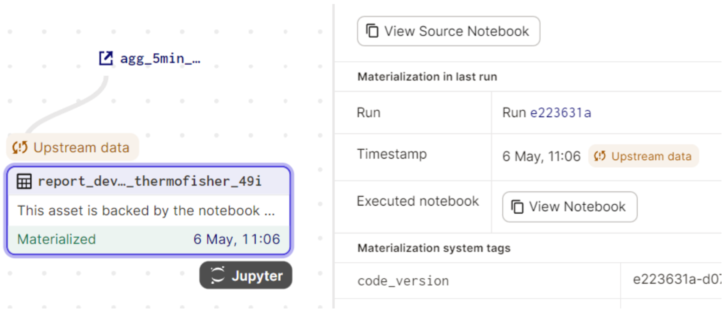
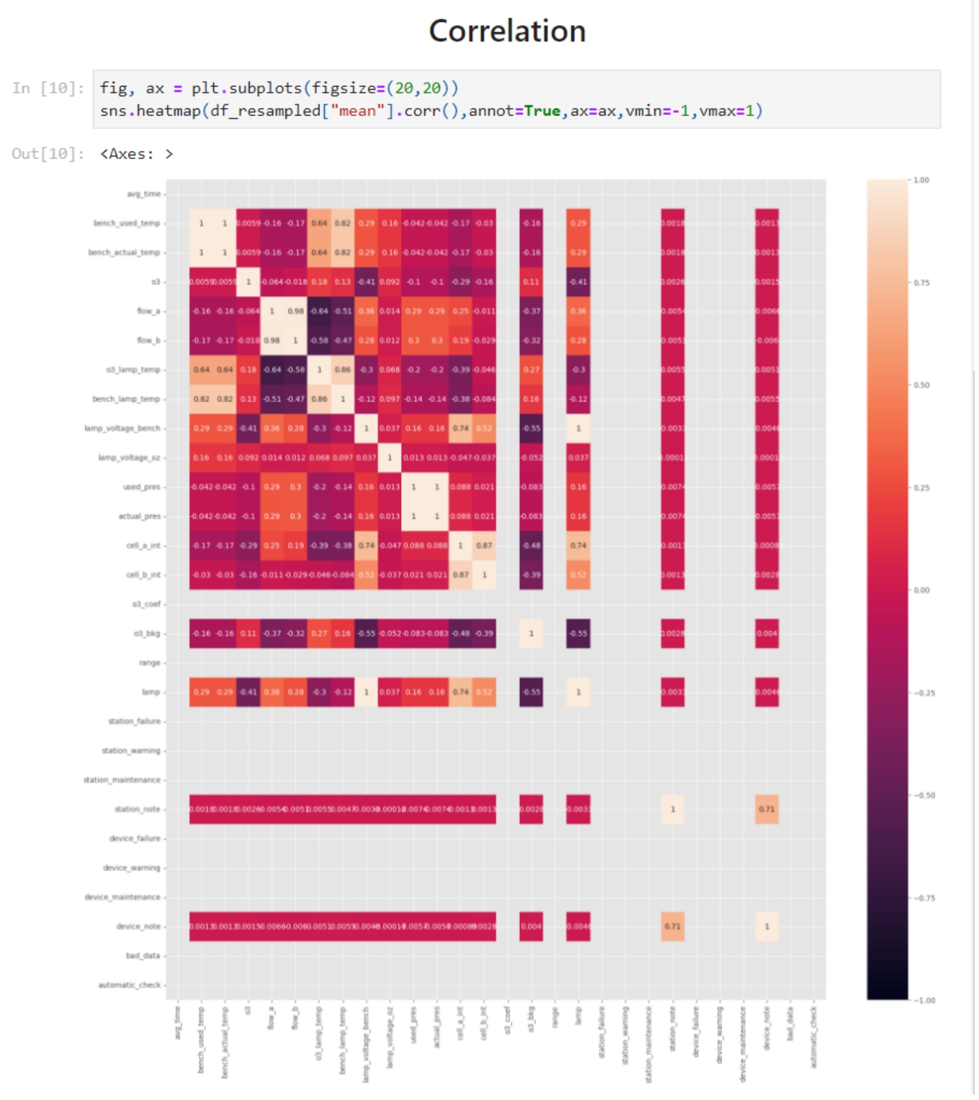


Figure 15. Dagit user interface view of the asset having the ability to see source and materialized notebook.

I suppose that one of the drawbacks of the currently deployed systems is that reports are generated per template, so if some of the imported libraries in one template were to change, we would have to manually modify the others. Since everything is programmed in a static manner, this also applies to the plotted visuals themselves. In the future, the Jupyter Templates library could be used to strictly manage the templates in order to make this process more convenient. This could also be included in the report's metadata to specify if a particular templating method was utilized for the generated asset.



Correlation

```
In [10]: fig, ax = plt.subplots(figsize=(20,20))
         sns.heatmap(df_resampled["mean"].corr(),annot=True,ax=ax,vmin=-1,vmax=1)

Out[10]: <Axes: >
```

Figure 16. Correlation plot between the measured variables used in the materialized report_device_thermofisher_49i asset notebook which uses the seaborn plotting library to display the correlation heatmap.

Users can easily share the asset (Figure 16) with the template to visualize data assets in their own unique ways, making the realization of the notebook generally successful for the environmental data assets and reports management system.

6 Future work

While developing this system, one of the concepts that occurred was the possibility to create a FAIR DOI integrated Dagster-based open research management system (foundation, 2023). Dagster appears to be a suitable fit for the criteria for such a system, considering the pre-existing tools and the way it works. Hereinafter is explained how the FAIR data principles (GFISCO, 2023) compared to those of Dagster:

Findable:

- F1. (Meta)data are assigned a globally unique and persistent identifier.
    - Given that Dagster requires each data asset to have a unique identity in order to function, this could be linked to the ability to register and link a DOI with each data asset.
- F2. Data are described with rich metadata (defined by R1 below)
    - Already implemented. Perhaps having the ability to use vocabularies to extend the functionality of readily describing data assets would be beneficial in a research environment.
- F3. Metadata clearly and explicitly include the identifier of the data they describe.
    - Already implemented via data validation. Given that Data validation rules must be satisfied for an asset to materialize, we can also verify that the data validation rules are true and that the data is in its proper format. Additionally, it is possible to describe data assets in free-form text to aid users in understanding the asset's contents.
- F4. (Meta)data are registered or indexed in a searchable resource.
    - If vocabulary-style metadata is implemented, it could be implemented as an external functionality to be able to validate that the metadata is in the corresponding format.

Accessible

- A1. (Meta)data are retrievable by their identifier using a standardised communication protocol.
    - Given that Dagster supports I/O Managers, it should be possible to implement a system that provides simple access to a necessary dataset.
- A1.1 The protocol is open, free, and universally implementable.

    - The Dagster project is currently open source so It would also be possible to keep the protocol up to the requirements. It would be beneficial to be able to use Dagster's built-in functionality to easily access Data assets from various DOIs and generate a Research Asset Lineage that illustrates how data assets are used across different research projects.
- A1.2 The protocol allows for an authentication and authorisation procedure, where necessary
    - Possible to implement. Dagster Cloud already implements Roles for users to administer, edit, or simply observe the lineage of data assets.
- A2. Metadata are accessible, even when the data are no longer available.
    - This already exists. Before materializing assets, it is possible to prototype and define what the data is intended to be together with metadata. If additional functionality was required, information regarding the asset's dematerialization could be included to inform users when the asset will no longer be accessible. Currently, it is quite helpful to determine how up-to-date the data asset is.

Interoperable

- I1. (Meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation.
    - Metadata has to stick to a strict format. However, if the data pipeline and its code were made public, we would be able to comprehend how the data is utilized without having to speculate.
- I2. (Meta)data use vocabularies that follow FAIR principles.
    - Like the earlier statement. Since we have data validation, we can establish strict guidelines for what makes valid data vocabulary. Additionally, we can always derive the assets required to follow vocabulary rules and define which rules they connect to.
- I3. (Meta)data include qualified references to other (meta)data.
    - Exists already in the form of Asset dependences. If it were possible to share data assets via DOIs and the asset provenance was accessible to all research projects, we would have a system to trace all data assets back to their origin together with metadata.

Reusable

- R1. (Meta)data are richly described with a plurality of accurate and relevant attributes

    - Already exists due to being able to describe metadata of a data asset. Dagster also helps with giving us access to the Data asset lineage to determine if the data is raw or not. Also with asset materialization we can see when the asset was generated.
- R1.1. (Meta)data are released with a clear and accessible data usage license.
    - Could be developed as a supplement to the DOI data access system. This could also enable us to set up asset lineage in order to generate exceptions when data is used in a non-licensed manner.
- R1.2. (Meta)data are associated with detailed provenance.
    - Already suitable for assets that rely on other assets. Raw data workflows could benefit from additional functionality to link how data is collected from non-data asset workflows.
- R1.3. (Meta)data meet domain-relevant community standards
    - Already incorporated into Dagster. In consideration of the fact that FAIR principles do not address quality assurance, we can say that Dagster is already ahead of the curve by offering data validation which in return would meet community standards.

Overall, Dagster would make it easier for researchers to delve into the work of other researchers to access the data assets used for the article or other derived assets. The articles could also be interactive in the sense that the data visualization and graphs were data assets. This would clarify which assets were utilized in every visualization.

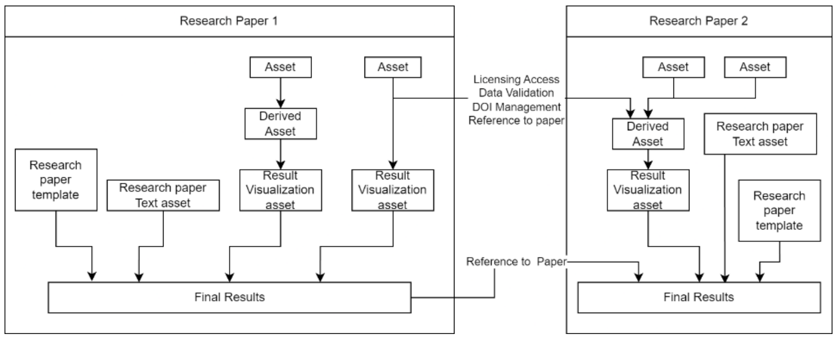


Figure 16. Basic overview of a Dagster usage in a research-based data management system.

It should also enable researchers to avoid dealing with extra data management steps such as data access, validation, versioning, and licensing issues if they are able to access assets from a governed system that incorporates Dagster. This could be accomplished by including code and data versioning for researchers to use. If the code was also made available to the public, we would be able to see how researchers utilized and transformed the data for future reference. Dagster can manage our I/O, so we could import datasets into our own project if we were able to link assets through

DOIs to allow researchers to not deal with file management but with the data itself. Figure 16 depicts how such a system could operate using fundamental concepts.

Conclusions

As a result of this project, a workflow for data management was proposed, implemented, and evaluated using a stack of open-source technology on environmental time series data for the Tahkuse measurement station. The utilized solution is able to generate 27 different data assets from two gas analyzers and three logbooks. The assets are versioned and metadata enriched together with descriptions to allow for other users to understand what type of data they are using.

This approach was created with environmental data assets and data reports in mind, and it enables users to understand what data assets and how they are produced. As a result, the created environmental data management system is robust, scalable, and versionable, allowing for simple modification and maintenance by utilizing Dagster-provided code, data, and dependency versioning together with Git.

Considering this approach to working with data, the system can in the future readily support other resources from which data can be served, allowing more devices to be added to the Tahkuse measurement station environmental data asset management system, and always have an up-to-date data asset with a clear record of what data is used.

It was also theorized that such a system could theoretically work in a larger-scale research environment, allowing for a simpler to understand and open research management system and providing researchers with an easier means to access data without having to deal with issues around the data management surrounding it, thereby allowing for more open-minded and connected possibilities for research.

References

Baker, M. (13. 9 2016. a.). *nature.com*. Allikas: Why scientists must share their research code: https://www.nature.com/articles/nature.2016.20504

Brown, A. W., Kathryn A. Kaiser , & David B. Allison. (2018). Issues with data and analyses: Errors, underlying themes, and potential solutions. *PNAS*. Allikas: https://www.pnas.org/doi/10.1073/pnas.1708279115#abstract

Dagster. (08. 05 2023. a.). *Create new project*. Allikas: Dagster Docs: https://docs.dagster.io/getting-started/create-new-project

Dagster. (01. 01 2023. a.). *Dagster main page*. Allikas: https://dagster.io

Dennis, C. (16. 1 2023. a.). *Hightouch*. Allikas: What is DuckDB: https://hightouch.com/blog/duckdb

foundation, d. (2023). *DOI*. Allikas: what is DOI: https://www.doi.org/

GFISCO. (2023). *FAIR*. Allikas: FAIR Principles: https://www.go-fair.org/fair-principles/

Git. (2023). *git-scm*. Allikas: https://git-scm.com/

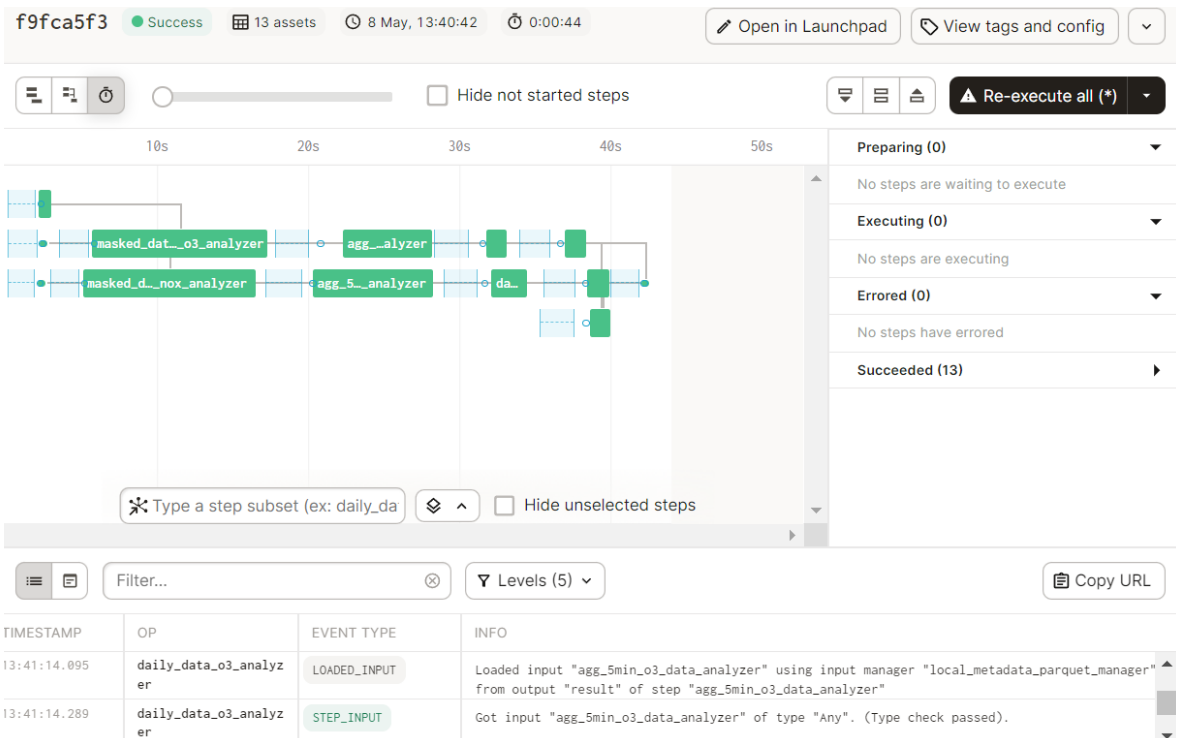
Jupyter. (2023). *Index*. Allikas: Jupyter : https://jupyter.org/

nteract, t. (2023). *Papermill*. Allikas: Docs: https://papermill.readthedocs.io/en/latest/

Union, E. (2023). Allikas: https://research-and-innovation.ec.europa.eu/strategy/strategy-2020-2024/our-digital-future/open-science_en#the-eus-open-science-policy

Appendix

I. Data asset materialization run graph view



TIMESTAMP	OP	EVENT TYPE	INFO
13:41:14.095	daily_data_o3_analyzer	LOADED_INPUT	Loaded input "agg_5min_o3_data_analyzer" using input manager "local_metadata_parquet_manager" from output "result" of step "agg_5min_o3_data_analyzer"
13:41:14.289	daily_data_o3_analyzer	STEP_INPUT	Got input "agg_5min_o3_data_analyzer" of type "Any". (Type check passed).

II. License

Non-exclusive licence to reproduce thesis and make thesis public