



Department of Management Science & Technology

MSc in Business Analytics

“Reproducibility Dashboards: Power BI vs. Python Dash”

By

Koutsogianni Agapi

Student ID Number: f2822316

Name of Supervisor: Charis Papageorgiou

July 2025

Athens, Greece



ACKNOWLEDGEMENTS

I would like to sincerely thank Professor Haris Papageorgiou for valuable guidance, encouragement, and continuous support during the development of this thesis. I'm also very grateful to Petros Stavropoulos for his practical assistance and helpful suggestions. Their advice and feedback played a key role in helping me stay focused and improve the quality of my work.

I also want to thank my family and friends for always being there for me. Their encouragement and understanding made a big difference during this demanding period.

Lastly, I would like to dedicate this thesis to the memory of my beloved grandmother, whom I sadly lost during the writing of this work. Her love, strength, and quiet support continue to guide and inspire me.



ABSTRACT

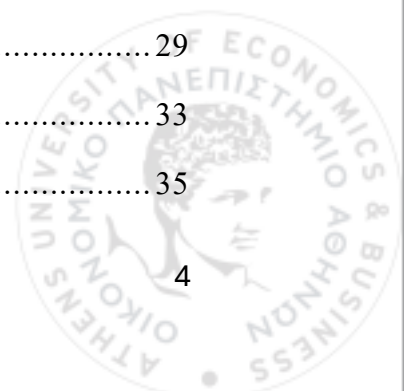
This thesis compares two dashboarding approaches—Power BI and Python Dash—for visualizing research evaluation data. The project involved creating interactive dashboards with both tools to help stakeholders monitor key performance indicators, research impact, and reproducibility metrics. Power BI provided an easy-to-use, low-code platform with polished visuals and simple sharing options, making it well-suited for public dashboards and business users. On the other hand, Python Dash took more development time but offered greater flexibility, modularity, and reproducibility, allowing for easier updates and integration with machine learning workflows.

The evaluation framework covered visual quality, interactivity, development effort, performance, scalability, and stakeholder communication. In the Python version, reusable components and centralized configuration logic (e.g. COLUMN_MAP) were created to support adaptability and version control. Visual and functional consistency was kept across all tabs, with custom features such as filter reset buttons and dynamic KPIs showing Dash's flexibility. The results show that while Power BI is better for quick, shareable business reports, Python Dash is ideal for research environments that need customization, traceability, and long-term maintenance.



CONTENTS

1. Introduction	8
1.1 Context & Motivation	8
1.2 Problem Statement	9
1.3 Objective	9
1.4 Scope	10
2. Literature Review	11
2.1 Concepts of Dashboarding	11
2.2 An overview of Power BI	11
2.3 An overview of dashboards based on Python	12
2.4 Comparative Studies	12
3. Dashboarding Tools & Setup	13
3.1 Tools	13
3.2 Dataset Description	15
3.3 Preprocessing Approach	17
3.4 Criteria for Evaluation	17
3.5 Approach	18
4. Methodology	20
4.1 Power BI Dashboard	20
4.1.1 General Info Tab	21
4.1.2 Overview Tab	22
4.1.3 Artefacts tab	24
4.1.4 Organizations Tab	25
4.1.5 Geo Tab	28
4.1.6 Trends Tab	29
4.2 Python Dashboard Development	33
4.2.3 Python Dashboard – Overview Tab	35



4.2.3 Python Dashboard – Artefact Tab	39
4.2.4 Python Dashboard – Organization Tab	41
4.2.6 Python Dashboard – Geographic Tab	43
4.2.7 Python Dashboard – Trends Tab.....	46
5. Comparison & Evaluation	51
5.1 Comparison Table	51
5.2 Design and User Experience	61
5.3 Ease of Development.....	62
5.4 Scalability and Maintainability	64
5.5 Code Reusability	66
5.6 User Experience and Stakeholder Communication	67
5.7 Reusability and template development	69
5.8 Performance and Loading Efficiency	70
5.9 Interactivity and Filtering Logic	72
5.10 Integration with External Tools and Pipelines	74
6. Discussion	76
6.1 Advantages and Limitations of Each Tool	76
6.2 Tool Suitability - Ideal Use Cases	77
6.3 Insights and Challenges During Implementation	78
7. Conclusion & Recommendations	80
7.1 Summary of Findings	80
7.2 Recommendation: When to Use Power BI vs. Python Dash	81
7.3 Suggestions for Future Work	81
8. References	84



FIGURES

Figure 1: Screenshot of the General Info tab displaying metadata cards and navigation button.....	21
Figure 2: Overview Tab of the Power BI Dashboard	23
Figure 3: Artefacts Tab of the Power BI Dashboard	25
Figure 4: Organizations Tab of the Power BI Dashboard	27
Figure 5: Geo Tab of the Power BI Dashboard	29
Figure 6: Trends Tab of the Power BI Dashboard	32
Figure 7: Overview Tab of Python Dashboard.....	38
Figure 8: Artefact Tab of Python Dashboard	41
Figure 9: Organization Tab of Python Dashboard.....	43
Figure 10: Geographic Tab of Python Dashboard	45
Figure 11: Trends Tab of Python Dashboard	49
Figure 12: Power BI KPI cards with refined layout.....	54
Figure 13: Dash KPI cards with dynamic updates	54
Figure 14: Power BI stacked bar chart of top 10 organizations by artefacts.....	55
Figure 15: Top 10 Organizations by Number of Artefacts.....	55
Figure 16: Number of Artefacts Over Time	56
Figure 17: Artefact Publication Trends by Category	56
Figure 18: Year-over-Year Metrics by Field of Science	57
Figure 19: YoY Growth of Citances and Reproducibility Metrics by Field of Science.....	57
Figure 20: Top 5 Artefacts by Reproducibility Composite Confidence Index (RCCI)	58
Figure 21: Geographic Distribution of Artefacts Across Europe	59
Figure 22: Global View of Artefact Distribution by Country	60

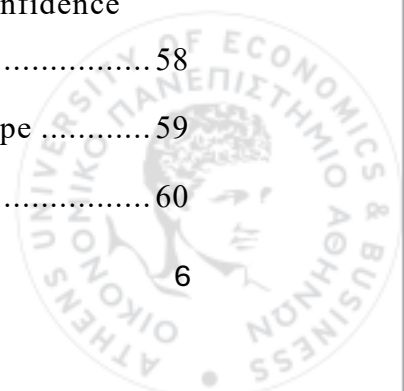


Figure 23: Dashboard Navigation Tabs	60
Figure 24: Section Navigation Tabs	61

TABLES

Table 1: Technical and Functional Comparison of Power BI and Python Dash	52
Table 2: Tool Recommendation Based on Project Scenario	77
Table 3: Recommended Tool Selection Based on Long-Term Objectives	81



1. Introduction

1.1 Context & Motivation

In the framework of the European research ecosystem, significant public funds are allotted to promote scientific progress, technological innovation, and societal impact, especially through programs like Horizon Europe. Funding organizations are depending more and more on data-driven technologies, including dashboards, to track results in order to guarantee accountability and increase return on investment. In particular, tracking the effect, reusability, and reproducibility of research artefacts (RAs), such as datasets, software, and techniques, is greatly aided by interactive and visible dashboards. These artifacts are important markers of a project's contribution to open science and information sharing because they are concrete results of financed research.

The demand for dashboards that are not just educational and aesthetically pleasing but also flexible and reusable across shifting datasets and institutional contexts is growing along with the volume and complexity of data. Accordingly, visualization platforms need to facilitate not only communication but also the changing requirements of reproducibility, data traceability, and user involvement, particularly in the context of research assessment frameworks.



1.2 Problem Statement

Choosing the best visualization platform is still a difficult task, even with the growing usage of dashboards in research reporting. Tools like Power BI are perfect for communicating information with stakeholders who are not experts since they offer polished images and quick development with few technical obstacles. Programming-based frameworks, like as Python's Plotly Dash, on the other hand, have a higher learning curve but provide better flexibility, extensibility, and integration with analytical workflows. In research settings, where information and measurements are constantly changing, the trade-off between flexibility and usability becomes especially important. Therefore, funders and analysts looking to create long-lasting, flexible dashboarding solutions must have a thorough awareness of each tool's advantages and disadvantages.

1.3 Objective

This thesis presents a comparative study of two dashboarding platforms—Power BI and Python (Plotly Dash)—by developing parallel dashboards that visualize key indicators related to the reusability, reproducibility, and impact of research artefacts generated in European Commission (EC)-funded projects. Finding the technology that best enables reusable, maintainable dashboards that can adapt to changes in data format and indicators over time is the major goal. The dashboard is designed to assist funding agencies in determining if the results of their investments are reproducible and scientifically sound.



1.4 Scope

The scope of this study is confined to the visualization and interaction layer of the dashboarding process. Backend data engineering tasks such as scraping or API-based data ingestion are excluded. Both dashboards are built using a structured Excel file as the main data source, which includes EC project metadata, citation and reuse metrics, artefact classifications, and reproducibility indicators such as FWCI, FWRI, FAIR Index, and Reproducibility Composite Confidence Index.

The evaluation focuses on key dashboarding aspects, including user interactivity, layout clarity, reusability of design and logic, flexibility to accommodate schema changes, and suitability for ongoing research monitoring. Usability is assessed from the perspective of stakeholders such as funding organizations and analysts, while adaptability is analyzed in terms of ease of maintenance, modularity, and support for future extension.

In order to place the dashboarding tools in a broader analytical and technological context, the next section reviews existing literature and best practices related to dashboard development, with a particular focus on visual communication, modular architecture, and reproducibility in research data visualization.



2. Literature Review

2.1 Concepts of Dashboarding

Dashboards are graphical user interfaces created to effectively communicate important metrics and trends. Few highlights that effective dashboards should provide simplicity, clarity, and real-time relevance to ensure users can analyze KPIs without distraction [1]. Both interpretability and user engagement are improved by using clear metrics, relevant visuals, and interactive features like filtering or time-based slicing. Knaflic further promotes data storytelling, using visual and narrative elements to transform raw data into insightful messages [2]. These guidelines are essential when communicating with non-technical stakeholders, such as funders or policymakers, about the reuse potential of complex research outputs.

2.2 An overview of Power BI

Microsoft created Power BI as a commercial business intelligence and analytics tool. Its drag-and-drop interface, integration with SQL, Azure, and Excel, and quick deployment make it popular among business users [3]. Because of its low-code nature, Power BI is widely adopted for its ease of use and secure sharing within the Microsoft ecosystem. However, limitations exist in terms of automation, advanced logic, and customization. Its closed-source architecture makes it harder to audit or adapt in open science environments [4].



2.3 An overview of dashboards based on Python

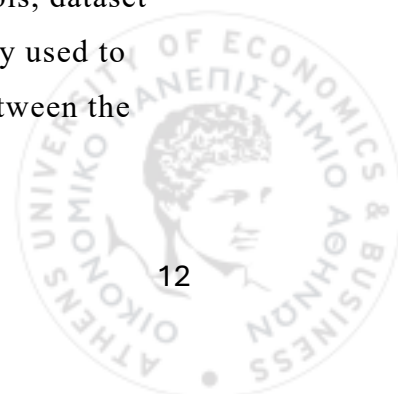
Python dashboards—built with tools such as Panel, Streamlit, and Plotly Dash—provide fully customizable, code-based alternatives. Dash allows developers to use Python for building rich, interactive dashboards while embedding logic, analytics pipelines, and complex callbacks directly in the interface [5]. These tools enable version control, modular development, and integration with machine learning models, which is particularly valuable in evolving research contexts. However, the learning curve and technical requirements are higher compared to Power BI [6].

2.4 Comparative Studies

Recent studies have highlighted the trade-offs between low-code platforms and open-source frameworks. While Power BI enables faster prototyping and broader user adoption, Python-based dashboards offer more flexibility and stronger alignment with reproducibility and FAIR data principles [4].

Ergasheva et al. highlight that open-source tools support the development of adaptable dashboards that can accommodate evolving data requirements and complex application metrics [6]. These findings are highly relevant for projects that prioritize transparency and long-term reusability.

This study takes a hands-on approach by developing two concurrent versions of a reproducibility monitoring dashboard, one using Power BI and the other using Python, in order to convert the theoretical insights acquired in the literature review into a useful, comparative analysis. In a research setting, these implementations form the basis for assessing the practicality of low-code versus code-first dashboarding paradigms. The technical tools, dataset properties, preprocessing techniques, and implementation strategy used to guarantee an impartial, consistent, and perceptive comparison between the two platforms are described in the next chapter



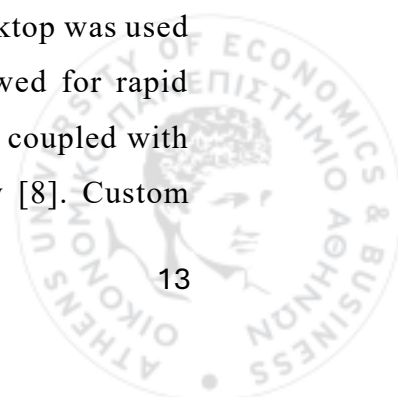
3. Dashboarding Tools & Setup

To explore how different dashboarding platforms perform in the context of evaluating research artefacts, this study adopts a structured, comparative approach. Two distinct technologies—Power BI and Python Dash—were selected to develop parallel implementations of a dashboard designed to monitor reproducibility and reuse indicators. Each tool offers unique strengths in terms of usability, flexibility, and alignment with open science practices. This section outlines the technical setup used in the project, including the tools selected, the dataset’s structure, preprocessing strategy, and the rationale for adopting a matched implementation methodology.

3.1 Tools

To evaluate and contrast the effectiveness of different dashboarding technologies in a research evaluation context, this study employed two distinct tools: Power BI and Python. Both were used to develop parallel implementations of a reproducibility-focused monitoring dashboard. The goal was to maintain as much functional parity as possible between the two versions while exploring their respective strengths, limitations, and suitability for template-based deployments. The Python version was developed entirely within Visual Studio Code (VS Code), which also served as the main environment for testing, version control, and code organization.

Power BI, developed by Microsoft, is a commercial business intelligence platform that has gained widespread adoption in both enterprise and academic settings. It provides a highly accessible, user-friendly interface for building data visualizations and reports [7]. For this project, Power BI Desktop was used to design and implement the dashboard [7]. The interface allowed for rapid development via drag-and-drop configuration of visual elements, coupled with powerful data transformation capabilities through Power Query [8]. Custom

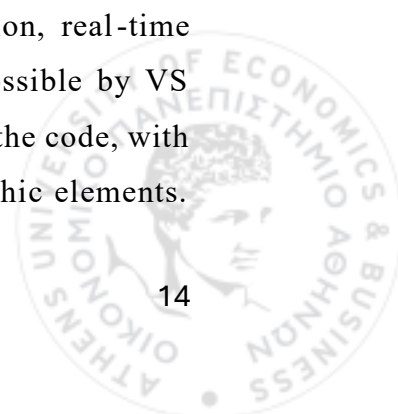


metrics and interactivity were defined using DAX (Data Analysis Expressions), enabling the creation of calculated fields, KPIs, and conditional logic within visuals [9]. Power BI's seamless integration with other Microsoft services made it especially appealing for stakeholders looking for straightforward deployment and web-based access via the Power BI Service [7].

The second version of the dashboard was implemented using Python, leveraging a modular combination of open-source libraries that enabled full control over both data processing and visualization logic. Dash, a Python framework created by Plotly for making interactive, web-based dashboards, was used to create the application [10]. In a code-first setting, it made it possible to seamlessly combine layout, interaction, and user-driven filtering. Dynamic features including donut charts, grouped and stacked bar charts, line graphs, pie charts, and treemaps were built using Plotly Express and Graph Objects [11]. Scrollable and filterable tabular data was displayed using Dash DataTable [10], and a responsive, aesthetically pleasing design was guaranteed by Dash Bootstrap Components [12]. Git version management was made possible by the development environment, VS Code, which promoted a neat, modular project structure [14].

Pandas, a popular Python data analysis library, was utilized for data processing and transformation. Reading structured Excel spreadsheets, cleaning and filtering them, and converting raw information into the formats needed for dashboard generation were all done with Pandas [13]. Dash Bootstrap Components offered grid systems, tabs, and stylized cards that emulated Power BI's elegant layout, ensuring responsiveness and layout consistency [12]. Lastly, Dash DataTable provided interactive, scrollable tables that allowed visitors to see research artefacts together with reproducibility scores and citation counts [10].

Visual Studio Code, which provided various benefits in terms of productivity and maintainability, was chosen for development. Git integration, real-time error checking, and efficient script development were made possible by VS Code [14]. A modular, library-style structure was used to arrange the code, with separate folders for layouts, callbacks, utility methods, and graphic elements.



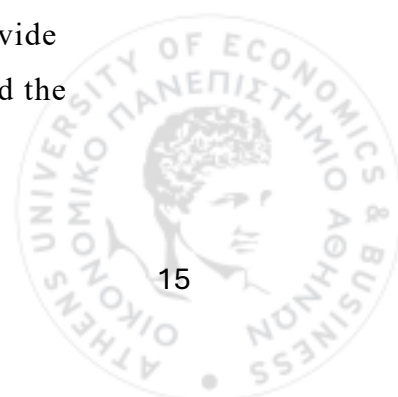
This architecture improved the dashboard's adaptability to future datasets or study fields, facilitated scalable development, and made debugging easier.

Together, these two implementations offered a comprehensive foundation for examine the trade-offs between low-code (Power BI) and code-first (Python Dash) dashboarding paradigms in the context of research monitoring and reproducibility assessment.

3.2 Dataset Description

The dataset used in this thesis originates from the Excel file developed for the Reproducibility Monitoring Dashboard initiative. This dashboard is part of a broader effort to evaluate the impact, reusability, and reproducibility of research artefacts funded by the European Commission (EC) under programs such as Horizon Europe. The data was provided in a structured Microsoft Excel (.xlsm) file [22], specifically within a sheet labeled “Sheet_tier.” To ensure a consistent basis for comparison, the dataset was preprocessed once and used in the same format for both the Power BI and Python dashboard implementations. Preprocessing steps included renaming columns for clarity, ensuring uniform data types (e.g., converting year fields to integers), and handling missing or incomplete values.

The dataset itself is rich and multifaceted, composed of several distinct but interconnected categories of information. The first section, spanning columns A through M, contains project and document metadata. This includes fields such as Report ID, Production and Published Dates, the reporting mode (e.g., whether the submission was a publication or data management plan), and detailed information about the creator and affiliated organization, including IDs, names, locations, and type (e.g., research organization, civil society organization, government body, or industry). These elements provide essential context for understanding the origins of the artefacts and the institutions producing them.



Columns N through Q describe the research artefacts themselves. Each row corresponds to a specific artefact—dataset, software tool, or methodological output—and includes Artefact ID, category and subcategory, and artefact name. Citation and mention metrics in columns R and S capture the number of citances (mentions in other texts) and formal citations [20]. These indicators serve as proxies for visibility and scholarly impact.

Further technical details are provided in columns Z through AF, covering artefact-specific metadata: scope (reused vs. originally created), license type, versioning information, and URL, which collectively contribute to traceability and accessibility. This section also records the nature of citances—supporting, refuting, or neutral—enabling the computation of a Reproducibility Confidence Index based on citation tone classification [19].

Scientific classification is addressed in columns AG and AH, where each artefact is categorized according to Field of Science (FOS) using the SciNoBo tool [17]. This dual-level classification (Levels 3 and 4) enables domain-specific analyses and more granular assessments across disciplines.

Finally, columns T through Y contain the core performance indicators: Field-Weighted Citation Impact (FWCI) [18], Field-Weighted Reusability Index (FWRI), a composite Reusability Index ($FWCI \times FWRI$), and a FAIR Index evaluating adherence to FAIR data principles (Findability, Accessibility, Interoperability, Reusability) [16]. A Reproducibility Composite Confidence Index combines the FAIR, reusability, and confidence metrics into a single holistic measure.

Taken together, these structured indicators offer a comprehensive framework for evaluating the scientific relevance, openness, and reusability of publicly funded research. The dataset was well-suited for the development of evaluation dashboards in both Power BI and Python, providing a shared foundation for visualizing key performance metrics and enabling strategic, data-driven decision-making by funders.



3.3 Preprocessing Approach

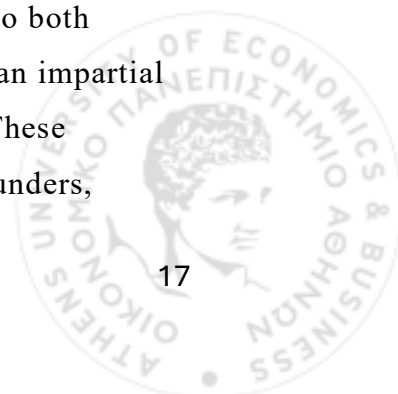
To ensure a fair and consistent comparison between the Power BI and Python implementations, the dataset underwent a single round of preprocessing and was subsequently used in the same format across both platforms. The preprocessing phase involved several critical steps. First, column names were renamed to enhance clarity and maintain consistency across the dashboarding environments. Next, appropriate data types were enforced—for example, converting year fields to integers—to ensure compatibility with filtering and time-series visualizations. Missing or incomplete values were also systematically handled to avoid runtime errors or misrepresentations in the visual output [23].

In the Python implementation specifically, an additional layer of abstraction was introduced through the creation of a configuration dictionary known as `COLUMN_MAP`. This mapping file decouples the column references from the core dashboard logic, allowing for seamless adaptation to future versions of the dataset that may include renamed or reorganized fields. As a result, the Python-based dashboard remains robust and maintainable even as the data schema evolves.

Overall, the pre-processed dataset—comprising rich metadata, classification labels, and a wide array of performance indicators—provides a comprehensive foundation for constructing evaluation dashboards, supporting more informed and evidence-based investment strategies [27].

3.4 Criteria for Evaluation

The dashboards were assessed using a set of standards pertinent to both technical performance and user experience in order to guarantee an impartial and organized comparison between Power BI and Python Dash. These standards were chosen to take into account the requirements of funders,

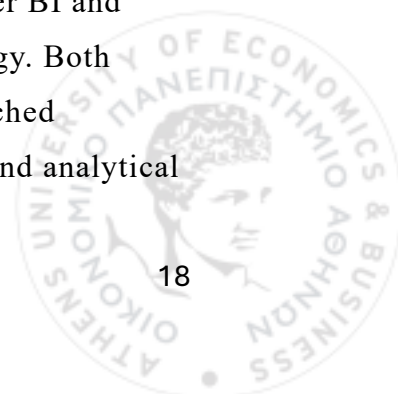


researchers, and data specialists who deal with dynamic and impactful datasets [24].

A systematic set of criteria that addressed both technical and user-centered aspects served as the basis for evaluating the two dashboards. Visual quality takes into account elements like visual hierarchy, layout consistency, font choice, and color balance to assess the interface's coherence, design refinement, and ability to communicate insights [25]. Interactivity evaluates the responsiveness of user inputs and how well the dashboards facilitate dynamic exploration using drill-down features, dropdown menus, tooltips, filters, and slicers [24]. Flexibility refers to the ability to customize dashboards to meet particular analytical requirements—such as changing layouts, creating unique metrics, or applying sophisticated formatting—and highlights the contrast between code-driven customization and graphical user interfaces. Development time and effort encompasses the time, technical expertise, and complexity required to create, update, and maintain the dashboards, as well as their accessibility for non-technical stakeholders [26]. Performance focuses on runtime characteristics—such as responsiveness to interactive elements, scalability as data volumes increase, and loading speed—that are essential for seamless end-user experiences [26]. Reproducibility measures how readily the dashboards' structure and logic can be tracked, versioned, shared, and reused in various contexts [27]. Finally, adaptability to changing data structures assesses each dashboard's ability to handle modifications in data formats and column schemas without requiring significant redesigns.

3.5 Approach

To ensure a controlled and meaningful comparison between Power BI and Python Dash, this study adopted a parallel implementation strategy. Both dashboarding tools were developed independently but under matched conditions: the same dataset was used for both, a similar visual and analytical



structure was followed, and identical evaluation criteria were applied [24]. This ensured that observed differences arose from the capabilities of each tool, rather than discrepancies in content or design logic.

The dashboards were structured around a shared set of thematic tabs—General Info, Overview, Artefacts, Organizations, Geo, and Trends—each offering a distinct analytical perspective on the dataset. Efforts were made to replicate similar visual elements such as KPI cards, radar charts, donut charts, stacked bar graphs, line plots, and interactive slicers [24]. A single preprocessed version of the dataset was used across both tools to ensure fairness and consistency. In Power BI, Power Query was used for data transformation, while in Python, data manipulation was performed using Pandas alongside a configuration-based abstraction layer (COLUMN_MAP), which maintained naming consistency and adaptability [23]. Although each implementation utilized the native features of its respective environment—such as DAX formulas and built-in visuals in Power BI, and Plotly charts with Dash callbacks in Python—the evaluation framework remained constant [24].

Following the development of both dashboards, a structured comparison was conducted to identify differences in development effort, customization capabilities, performance with complex datasets, and overall suitability for long-term research monitoring. This parallel strategy ensured that the findings were grounded in direct implementation experience, thereby providing an objective foundation for the comparative analysis presented in Section 5.

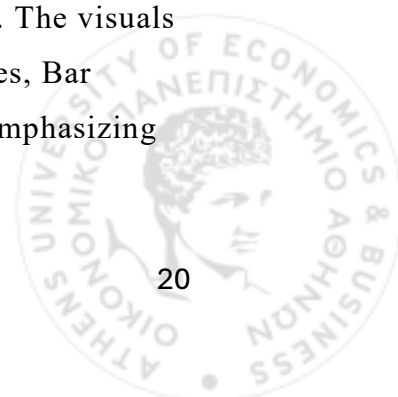
4. Methodology

This section describes the process used to create, build, and evaluate interactive dashboards with Python Dash and Power BI. Data pretreatment, dashboard design, and visual analytics implementation on both platforms are all part of the strategy. In order to guarantee a fair comparison in terms of usability, visual effectiveness, adaptability, and repeatability, emphasis is given on replicating features. The technique also describes the modular design of the Python dashboard and the choice of visualizations and key performance indicators (KPIs) that are pertinent to the target audience, which consists of investors monitoring the impact of research funding.

4.1 Power BI Dashboard

The Power BI dashboard was developed as the primary interface for visualizing and exploring research-related data. It connects to an Excel data source composed of two key sheets: *Info*, which contains contextual project-level metadata (e.g., organization, timeframe, description), and *Sheet_Tier*, which includes detailed artefact-level metrics [22]. The dashboard is designed to provide both a high-level overview and detailed drill-down capabilities, making it accessible for both strategic and analytical use cases [3]. The navigation is organized into multiple tabs—General Info, Overview, Artefacts, Organizations, Geo, and Trends—each offering different perspectives on the underlying dataset [29].

Each tab is designed with both interactivity and clarity in mind. Filters applied in one view persist across others, allowing users to maintain their analytical context while exploring different facets of the data [8]. The visuals rely heavily on native Power BI components such as Cards, Tables, Bar Charts, Maps, and Drill-through Buttons, with layout decisions emphasizing clarity, readability, and quick insight extraction [9].



4.1.1 General Info Tab

The Power BI dashboard is accessed through the General Info page, which provides users with a brief synopsis of the organization under study. All displayed information is dynamically pulled from the source Excel file's *Info* sheet. The organization name, the project duration, the list of related project IDs, and a brief textual description of the research activity are the four main data items presented in the tab's neat, card-based structure. Card visualizations are used to display these values, each directly bound to its corresponding column (e.g., "Organization," "Period," "Projects," "Description") [22].

The layout clearly communicates context before users proceed to deeper analytical sections. A "Go to Dashboard" button at the bottom of the tab enables quick navigation to the primary analytical views. Overall, the General Info tab anchors users by providing consistent metadata drawn directly from the structured data source [7].

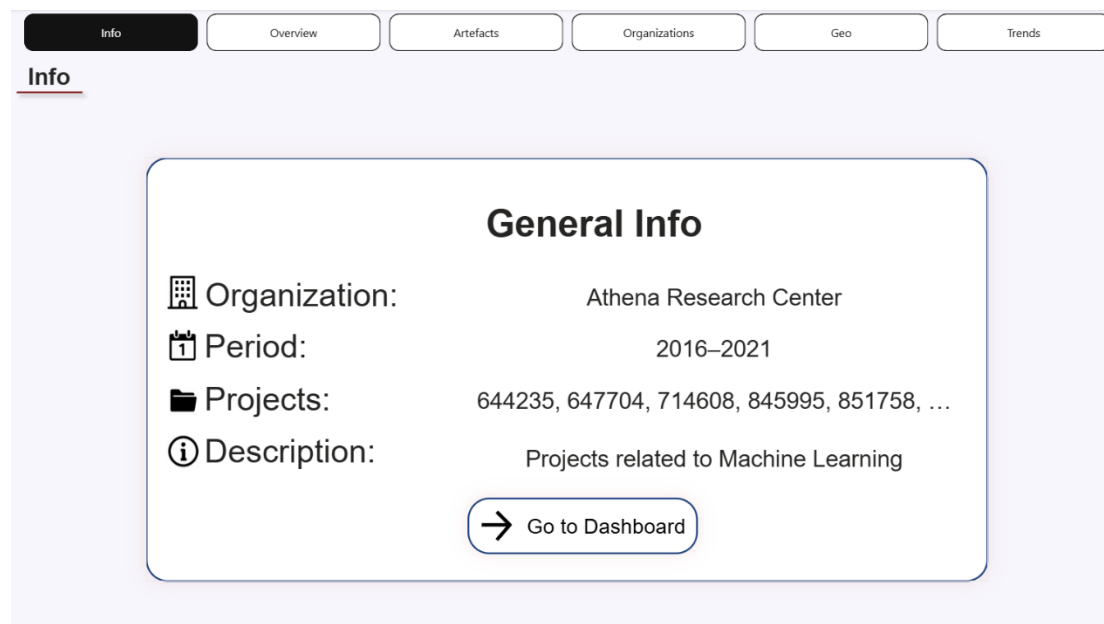
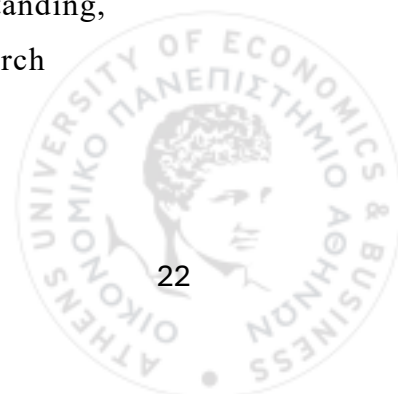


Figure 1: Screenshot of the General Info tab displaying metadata cards and navigation button.

4.1.2 Overview Tab

The Power BI dashboard serves as the original implementation for visualizing the dataset and tracking project-level metrics. It is integrated into Power BI Desktop and establishes a direct connection to an Excel file that has two main sheets: `sheet_tier` and `info`. The dashboard was created to offer a clear, all-inclusive interface for examining important characteristics of research artefacts such as impact, reproducibility, and classification, across various organizations and time periods. From general institutional framework to artefact distributions and organizational patterns, each tab represents a different analytical scope. A well-defined tab bar facilitates section navigation, enabling readers to navigate between the report's various pages with ease. Standardized iconography, a unified color scheme, and consistent visual formatting all help to create a cohesive user experience that improves usability and appearance. Interactive slicers positioned throughout all tabs enable dynamic filtering, enabling data exploration according to dimensions such as organization, artefact type, research category, and year.

The Overview tab offers a high-level overview of the dataset by integrating interactive tools, filters, visualizations, and KPIs into a single, cohesive view. Aggregated metrics like Citances, Citations, FWCI (Field-Weighted Citation Impact), FWRI (Field-Weighted Reproducibility Impact), RI (Reusability Index), FI (FAIRness Index), RCI (Reproducibility Confidence Index), and RCCI (Reproducibility Composite Confidence Index) are shown on a row of KPI cards at the top of the page. Every card has rounded corners, faint drop shadows, and a uniform color scheme, with blue for repeatability indicators like FWRI and scarlet for citation-based measures like FWCI. The cards contain embedded information icons that, when hovered over, reveal comprehensive explanations and calculation logic (such as how FWCI is calculated). This guarantees openness and facilitates user understanding, especially for audiences that are not familiar with technical research measurements.



Users can dynamically filter the data by choosing Organization, Type, Category, Year, and a custom Scope toggle using a series of interactive slicers located beneath the KPIs [8]. The scope selector, which is a binary toggle between “In” and “Out” on the left-hand sidebar, probably indicates whether artefacts are filtered according to another inclusion criterion or ascribed internally or externally [8]. A question mark icon for assistance or user guidance, a camera icon that launches the Windows screenshot tool (Win + Shift + S), and a PDF export icon for downloading the current view are additional sidebar icons that improve interactivity [2]. These features are especially helpful for presentations, stakeholder reporting, or documentation needs.



Figure 2: Overview Tab of the Power BI Dashboard

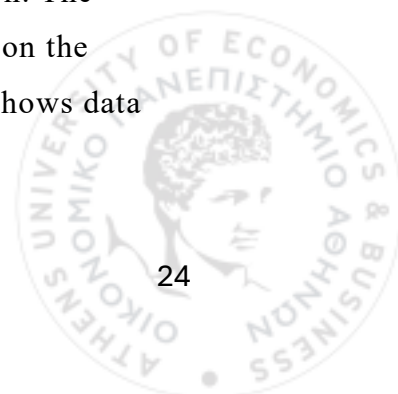
The lower section of the tab features six key visualizations. The percentage of artefacts in each category, such as software and dataset, is depicted via a donut chart. Whereas a treemap illustrates the distribution of artefacts across several Fields of Science (FoS), with tile size signifying the number of artefacts per FoS Level 3 categorization, a bar chart depicts the geographic distribution of artefacts by nation. Two grouped bar charts—one that compares average FWCI and FWRI and the other that shows average RI and RCCI—offer a more

thorough understanding of measure performance by category. All of these interactive graphics provide users with a thorough overview of the volume of artefacts, their geographic reach, their disciplinary distribution, and the quality of their impacts. They also immediately update in response to choices made on the slicer or scope. The Overview tab establishes the framework for more in-depth examination in the tabs that follow on the dashboard [2].

4.1.3 Artefacts tab

The Power BI dashboard's Artefacts tab moves the analytical focus from aggregate metrics to object-level evaluation by concentrating on the attributes and performance of individual research artefacts. This section delves into identifying exceptional artefacts based on their reusability and reproducibility scores, in contrast to the Overview tab, which provides a summary of trends across categories. A comprehensive data table including artefacts and important metrics like Citances, FWCI, FWRI, RI, RCI, and RCCI forms the layout's focal point [29]. Users can compare artefacts side by side and see how many indicators differ among them thanks to this tabular format. A horizontal bar chart with the top 5 artefacts ranked by Reusability Index (RI) on the left shows which have the most potential for reuse. The top 5 artefacts by RCCI are displayed in a treemap visualization, which provides a condensed visual representation of repeatability confidence.

Further, the tab includes a “Supported & Refuted Artefacts by FoS” graphic, which offers a more nuanced perspective on scientific influence. This visualization plots the number of supporting and refuting instances across Fields of Science, helping users assess which disciplines contribute most to validation or challenge of artefacts. This type of polarity analysis is exclusive to this tab and enhances the Overview's aggregate FoS distribution. The distribution of artefacts among nations is shown in a donut chart on the bottom right. Although it seems similar to other charts, this one shows data unique to each artefact rather than overall counts.



With synchronized dropdowns for Organization, Type, Category, and Year, filtering is still a crucial feature that enables focused delving into data at the artefact level. The vertically oriented Scope selection on the left still provides binary segmentation between artefact groupings (such as "In" vs. "Out"), which most likely correspond to internal versus exterior classifications. While improving accessibility during artefact exploration or reporting, supporting features like the help icon, screenshot capture, and PDF export are still available in the sidebar. All things considered, the Artefacts tab enhances the dashboard's structural logic while deepening analysis by enabling users to follow insights all the way down to the level of specific research findings.

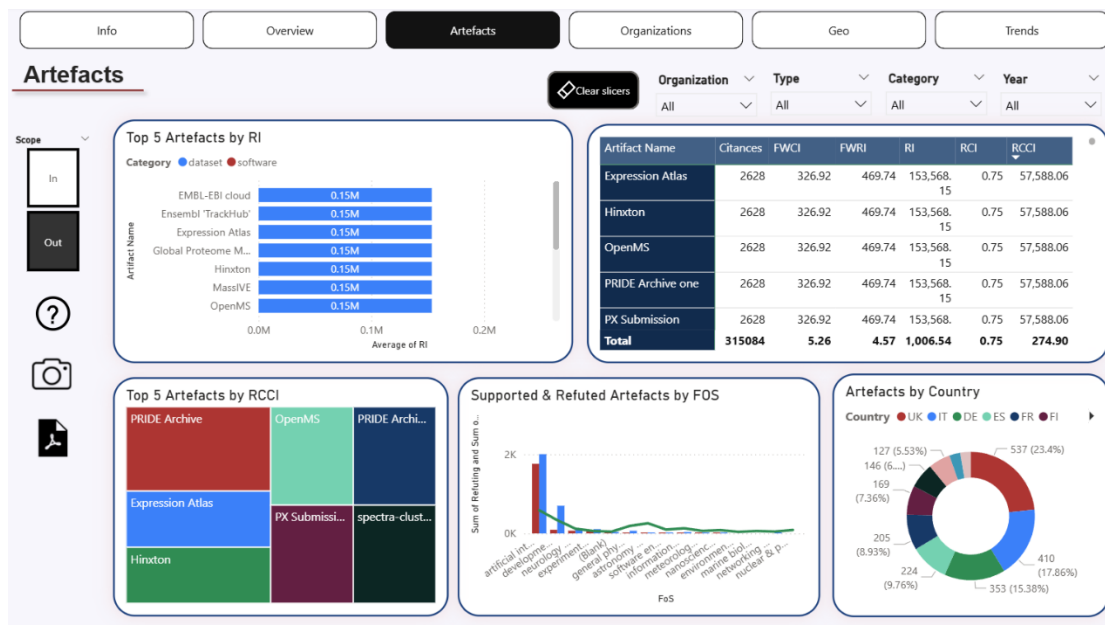


Figure 3: Artefacts Tab of the Power BI Dashboard

4.1.4 Organizations Tab

The *Organizations* tab of the Power BI dashboard offers a targeted examination of institutional contributions to research artefacts, emphasizing both the quantity and qualitative impact of outputs across different

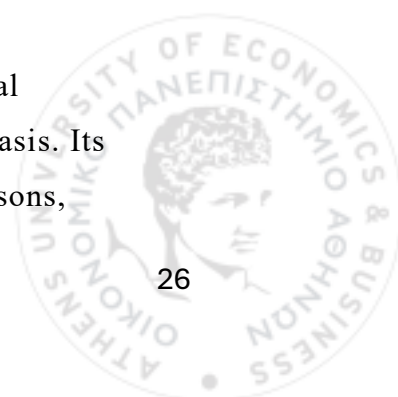
organizations. By shifting the analytical emphasis from artefact-level measurements to a higher-level institutional perspective, this view makes it possible to comprehend in depth which organizations are the most active and significant in the dataset.

A stacked bar chart with the Top 10 Organizations by Number of Artefacts, broken down by Field of Science (FoS), is displayed in the upper-left section. This graphic shows the disciplinary diversity of the contributions made by each institution in addition to quantifying their production. Segments that are color-coded help to differentiate between various scientific fields and provide instant insight into each organization's research focus. For example, although some institutions show expertise in particular topics like astrophysics or artificial intelligence, others may show a balanced distribution across several fields.

Adjacent to this, a data table ranks organizations by several key performance indicators: Citances, FWCI (Field-Weighted Citation Impact), FWRI (Field-Weighted Reproducibility Impact), RI (Reusability Index), RCI (Reproducibility Confidence Index), and RCCI (Reproducibility Composite Confidence Index). The table summarizes each institution's overall performance and production, enabling simple comparisons between them. To facilitate interpretation, a total row is included as a comparison to the aggregated dataset.

Two more bar charts focused on qualitative measures are added to the tab's bottom row. The institutions whose artefacts are most commonly reused are highlighted in the Top 5 Organizations per RI, demonstrating their wider application or utility. In contrast, the Top 5 Organizations per RCCI ranks organizations according to their average RCCI, which is a measure of repeatability strength. By shifting the focus from volume to quality, these charts help stakeholders find high-impact contributors who go beyond production alone.

By moving from measurements at the artefact level to institutional performance, the Organizations tab broadens the analytical emphasis. Its design places a strong emphasis on cross-organizational comparisons,



assisting users in identifying the top institutions in terms of production volume and qualitative impact. In order to identify top contributors and showcase research capabilities across many scientific fields, the visualizations place a strong priority on clarity and benchmarking. The tab maintains structural consistency with earlier pages, incorporating standard dropdown filters and a Scope toggle for customized exploration. Sidebar tools for exporting and user assistance are also retained, ensuring a smooth and unified interaction experience.

Overall, the Organizations tab offers a thorough and engaging interface for examining institutional performance. It allows users to assess the scientific relevance, reproducibility, and reusability of their outputs in addition to identifying important contributors to the creation of artefacts. Funders, legislators, or research coordinators who want to monitor institutional performance in a variety of ways will find this feature especially helpful.



Figure 4: Organizations Tab of the Power BI Dashboard

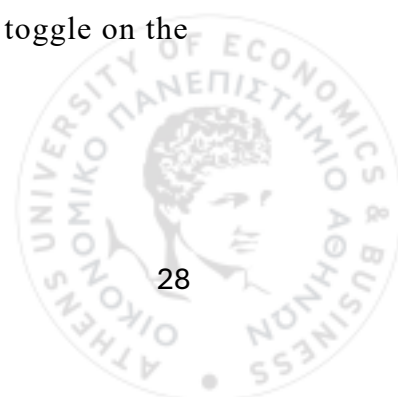


4.1.5 Geo Tab

The *Geo* tab of the Power BI dashboard introduces a spatial dimension to the analysis by visualizing the geographical distribution of research artefacts. Its central element is an interactive map titled “*Number of Artefacts per Country*”, which combines both the volume and quality of artefacts per nation. The map provides an easy-to-use method for identifying regional concentrations and relative impact by using bubble sizes to indicate the quantity of artefacts created and a color scale to represent the average Reusability Index (RI).

When hovering over each country, tooltips reveal detailed metrics such as the count of artefacts, average RI, and Reproducibility Composite Confidence Index (RCCI). For example, selecting Italy displays 49 artefacts, an average RI of 69.18, and an average RCCI of 13.14. This interactive feature allows users to examine national performance in terms of reproducibility and reuse potential. To the right of the map, a table lists countries with their respective values for core indicators including Citances, Field-Weighted Citation Impact (FWCI), Field-Weighted Reproducibility Impact (FWRI), RI (Reusability Index), Reproducibility Confidence Index (RCI), and RCCI (Reproducibility Composite Confidence Index). Below it, another matrix highlights top-performing artefacts, enabling direct comparison of citation and reproducibility scores at the artefact level.

The treemap visualization categorizes artefacts by Field of Science (FoS), offering insight into disciplinary focus by country. By highlighting the most active or significant study domains, this image enhances the geographic and artefact-level measurements. Users can dynamically filter the visual elements using slicers like Organization, Type, Category, and Year that are found at the top of the tab. These filters improve interaction and offer a multi-layered exploring experience when used in conjunction with the "Scope" toggle on the left-hand side, which alternates between "In" and "Out."



In conclusion, the Geo tab is essential for connecting analytical data with geographic context. It supports data-driven decisions and focused policy evaluation by allowing users to track performance in reproducibility and reusability, evaluate national contributions, and identify disciplinary strengths.

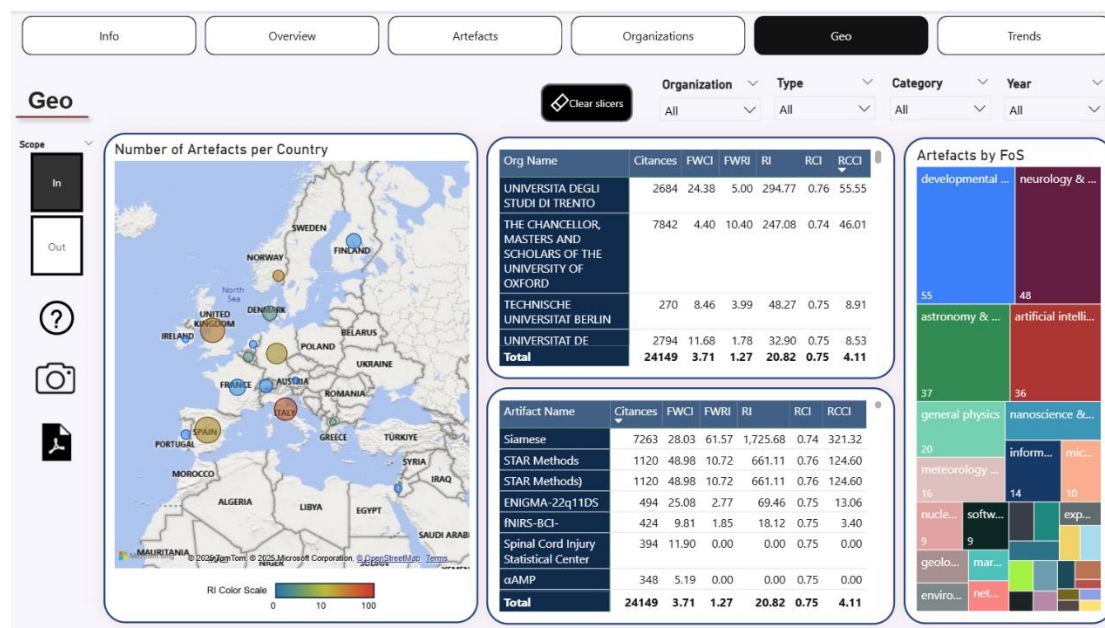


Figure 5: Geo Tab of the Power BI Dashboard

4.1.6 Trends Tab

The *Trends* tab offers a temporal analysis of the dataset, enabling users to explore how key metrics evolve over time across different Fields of Science (FoS). Through year-over-year (YoY) comparisons, it aims to draw attention to changes in the visibility, repeatability, and reuse of artefacts. Users can isolate and analyze changes within particular research contexts by using the tab's dynamic filters for Organization, Type, Category, and Year.

The tab's major feature is a comprehensive table that shows annual aggregated metrics by FoS. Citances, Citations, Reusability Index (RI), Reproducibility Confidence Index (RCI), and Reproducibility Composite Confidence Index

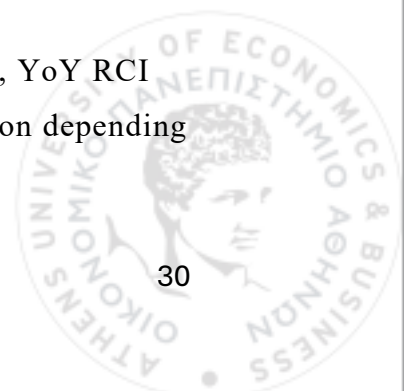
(RCCI) are all included, along with their absolute values and computed percentage changes. There is a YoY % gain or decline associated with each of these indicators. For each FoS, these YoY values are calculated using DAX metrics that compare the chosen year to the prior one. For example, the following script defines the YoY Citances % statistic as the percentage difference between the total Citances of the current year and the previous year, given the same FoS:

```

YoY Citances % =
VAR LastYearCitances =
    CALCULATE(SUM('Sheet_tier'[Citances]),
        FILTER(ALL('Sheet_tier'),
            'Sheet_tier'[Published Year] =
                SELECTEDVALUE('Sheet_tier'[Published Year]) - 1 &&
            'Sheet_tier'[FoS] = SELECTEDVALUE('Sheet_tier'[FoS])
        )
    )
VAR CurrentCitances = SUM('Sheet_tier'[Citances])
RETURN
    IF(ISBLANK(LastYearCitances), BLANK(),
        DIVIDE(CurrentCitances - LastYearCitances, LastYearCitances,
            BLANK())
    )

```

Similar logic is applied to calculate YoY Citations %, YoY RI %, YoY RCI %, and YoY RCCI %, using either SUM or AVERAGE aggregation depending



on the metric's nature. This allows for an accurate and scalable method of trend analysis, even as new data is appended to the dataset.

Three time-series charts are included beneath the table to further highlight longitudinal trends. The balance and development of scientific support or disagreement are depicted in the first area chart, which tracks the Sum of Supporting and Refuting Citances Over Time. This is particularly important when evaluating the reliability and significance of study findings. Plotting the Number of Artefacts Over Time by category (e.g., software vs. dataset) in the second line chart allows for the evaluation of production patterns. The third figure aids users in assessing repeatability performance over time by visualizing the RI and RCCI annual averages. A vertical year slicer on the right allows users to focus the tab's visuals on specific years. This interactivity, combined with the custom Scope toggle ("In" vs "Out"), provides a flexible exploration tool for both high-level trend analysis and targeted deep dives.

Overall, the Trends tab gives the dashboard more temporal granularity and acts as a diagnostic lens that allows stakeholders and academics to evaluate how impact and reproducibility measures have changed over time across different study fields. It completes the analytical framework by enhancing the categorical analyses in the Overview and Artefacts tabs and the spatial insights from the Geo tab. In particular, by adding a time-based component to artefact performance and scientific validation, the Trends tab expands upon the Artefacts tab. The Trends page allows for longitudinal monitoring of reuse, reproducibility, and support trends, allowing users to observe how these attributes change over time, whereas the Artefacts tab provides static, point-in-time evaluations.



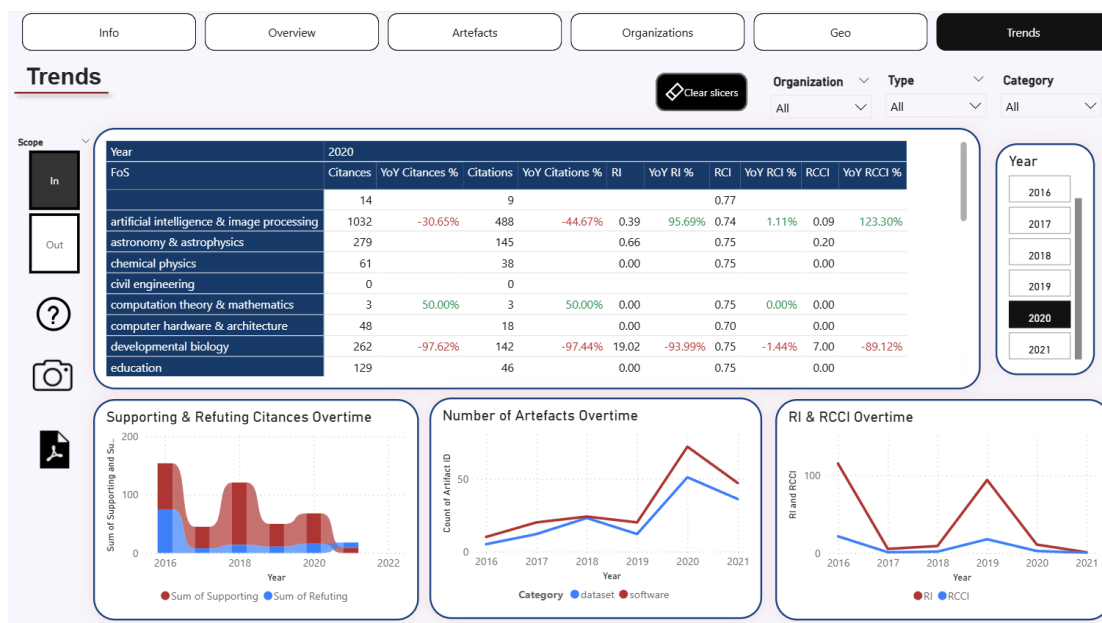


Figure 6: Trends Tab of the Power BI Dashboard

Although the Power BI solution provides a thorough and intuitive visualization of important indicators, it is limited by low-code environments, especially when it comes to extensive customization, reproducibility, and programmatic workflow interaction. In research contexts, where flexibility, modularity, and transparency are crucial, these limitations may present difficulties (Baars & Kemper, 2018; Alloghani et al., 2020).

An comparable dashboard was created using Plotly Dash, an open-source Python framework made especially for making interactive, web-based data visualizations, in order to overcome these constraints. Dash, as opposed to Power BI, gives complete control over the logic, structure, and behavior of dashboards, which makes it ideal for applications requiring interaction with dynamic datasets or scientific workflows (Sievert, 2020) [28].

Visual Studio Code (VS Code), a small yet effective code editor that facilitates Python programming, Git integration, and modular architecture, was used to create the dashboard. A more effective and repeatable development process was made possible by VS Code's facilitation of project management, debugging, and iterative testing. This development environment and Dash's adaptability made it possible to create a dashboard that is

completely reusable and adjustable, allowing it to adjust to future data upgrades without requiring manual interface reconfiguration [32].

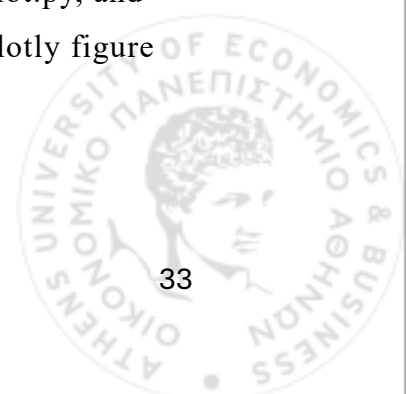
4.2 Python Dashboard Development

The Python-based dashboard was created in Visual Studio Code (VS Code) using a modular architecture and Plotly's Dash framework [30]. The main goal was to mimic the Power BI dashboard's capabilities while obtaining the benefits of code reuse, version control, and custom logic.

The Python-based dashboard follows a modular, library-style architecture, which ensures clarity, reusability, and scalability [31]. This structured approach promotes separation of concerns by assigning specific roles to dedicated folders, making the dashboard easier to maintain, extend, and adapt for future datasets or evolving user needs. The project is organized into clearly defined subfolders, each responsible for a distinct aspect of the application—ranging from layout and logic to data processing and styling.

At the core of the application lies the `app.py` file, which serves as the main entry point. It initializes the Dash app, registers all necessary callbacks, and integrates the layout components imported from the modular tab files. The `layouts/` folder contains individual layout definitions for each tab—such as `overview.py`, `geographic.py`, and `artefact.py`—constructed using Dash elements like `dcc.Graph`, `dash_table.DataTable`, and `dbc.Row`.

Interactive behavior is encapsulated in the `callbacks/` directory, where each file—such as `overview_callbacks.py` or `geo_callbacks.py`—defines Dash callback functions responsible for filtering data, updating charts, and responding to user input. Reusable visualizations are handled in the `graphs/` folder, which includes files like `line_chart.py`, `pie_chart.py`, `barplot.py`, and `stacked_bar_chart.py`, each containing a function that returns a Plotly figure tailored for a specific chart type.

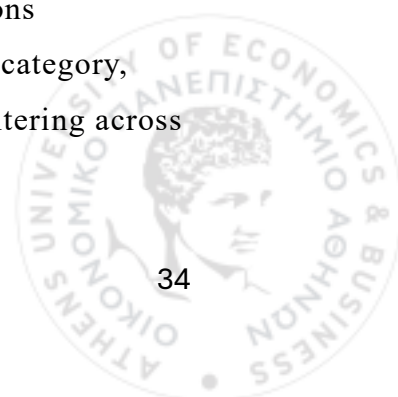


The components/ folder includes reusable layout elements such as dropdown-based selection boxes and KPI sections, with files like `kpi_section.py` and `selection_boxes.py`. To manage adaptability across datasets, the settings/ folder contains `column_config.py`, which defines the `COLUMN_MAP` dictionary—a key feature that decouples raw column names from the dashboard logic and enables compatibility with evolving data schemas.

Supporting scripts are placed in the `utils/` directory, which includes utility functions for loading and filtering data (`data_loader.py`, `data_filter.py`) as well as generating KPIs (`kpi_generator.py`). Lastly, the `assets/` folder houses a custom CSS file (`custom.css`), which provides a unified styling layer to ensure the dashboard maintains a clean and professional appearance. Together, this structured design not only enhances development efficiency but also reinforces the dashboard's role as a reusable and adaptable template for research evaluation. Code readability, reusability, and scalability are improved by this modular approach, particularly in research projects that may change or be transferred to new contributors. This dashboard's use of a centralized column abstraction layer is one of its main features.

This dictionary permits the use of logical names (such as "category") for all data references in the code, which are mapped to the Excel file's real column headers. Therefore, just this mapping needs to be changed, not the rest of the code, if a new dataset contains slightly different column names (for example, "Category Name" instead of "Category"). `config.py` also centralizes the file path (`DATA_FILE_PATH`), Sheet name (`SHEET_NAME`) and Dashboard title (`APP_TITLE`). This enhances future-proofing, portability, and modularity—all critical characteristics for research dashboards that could change or be used in various settings.

The core filtering logic resides in `utils/data_filter.py` and is invoked by various callback modules, such as `overview_callbacks.py` and `artefact_callbacks.py`. The utility function `filter_data_by_selections` dynamically segments the dataset based on user inputs including category, type, scope, organization, and year. This enables synchronized filtering across all dashboard tabs using standardized selection controls.



Because of this modular design, logic and layout can be clearly separated, making it simple to reuse components across different dashboard tabs. It maintains a consistent experience across the application by guaranteeing consistent behavior when users interact with filters. Furthermore, the implementation can be easily modified to accommodate future datasets by using the `COLUMN_MAP` to reference column names, which improves scalability and maintainability.

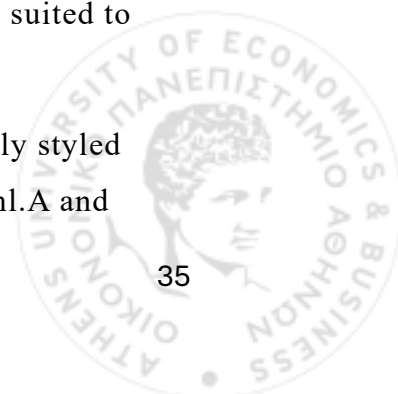
4.2.3 Python Dashboard – Overview Tab

Plotly's Dash framework was used to create the Python-based dashboard, which was designed to closely resemble the Power BI design while providing more flexibility and programmatic control. The dashboard's "Overview" tab, which features interactive elements and responsive visualizations, offers a comprehensive snapshot of key metrics related to citations, reproducibility, and research reuse.

The Overview tab's layout is defined by the file `layouts/overview.py`, which also employs a modular structure to organize the logic and graphics. The tab is made up of several dynamic elements, including interactive graphs, KPI indicators, and filters. Each of these sections is constructed as a reusable component for improved clarity and scalability.

The `selection_boxes()` function, defined in `components/selection_boxes.py`, creates the dropdown filters at the top of the tab. Users can utilize these filters to refine the data that is shown according to a number of parameters, such as Scope, Organization, Type, Category, and Year. Options are loaded from the dataset and labeled properly in each dropdown, which is designed for visual clarity and interactivity. Before any visualizations are displayed, this configuration guarantees that users can examine data subsets suited to their interests or organizational context.

Positioned next to the filters, the dashboard includes a prominently styled "General Info" button. This component is defined using Dash `html.A` and



dbc.Button, and when clicked, it opens a supplementary PDF document (reproducibility_dashboard.pdf) in a new browser tab. This feature provides immediate access to supporting documentation, helping users understand the metrics, methodology, and definitions behind the visualizations presented in the dashboard.

Immediately below the filters, a row of key performance indicator (KPI) cards offers users a quick summary of central metrics. Every KPI is presented on a neat, rounded card with a standardized font, color scheme, and space. The utility functions `generate_kpis()` and `generate_kpi_section()` are used to dynamically generate this section.

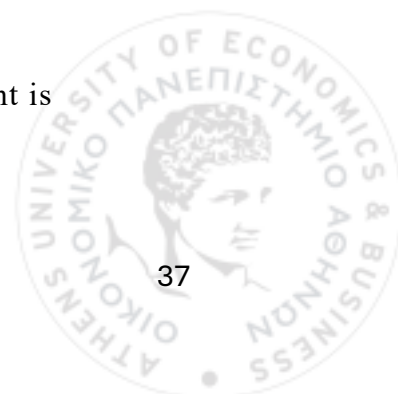
Citances, Citations, FWCI, FWRI, FI, RI, RCI, and RCCI are the eight different indicators that are displayed on the Overview tab. These indicators offer a succinct overview of the reproducibility, reuse, and impact of research. Every KPI is shown on a rounded, stylized card that is intended to be responsive and visually clear. A consistent and accessible user experience across devices is guaranteed by the layout. Users are constantly interacting with the most pertinent data context since all values are dynamically updated in real time based on active filter selections.

All graphic elements in the dashboard are housed within the `graphs/` directory, each modularized into its own Python script using Plotly Express to ensure consistent styling and reusability. The Overview tab features six key visualizations, thoughtfully divided into two horizontal rows to provide a clear and visually balanced layout. The first row includes a Pie Chart (`pie_chart.py`) that displays the distribution of citances by category, helping users distinguish between different types of research outputs such as Software or Dataset. Alongside it is a horizontal Barplot (`barplot.py`) showing the number of unique artefacts per country; this chart is interactive, allowing users to click on a bar to filter the dataset by country, with a “Clear country filter” button to revert the selection. Completing the first row is a Grouped Bar Chart (`grouped_barplot.py`) that compares FWRI and FWCI metrics by category, offering two complementary perspectives on research reuse and impact.

The second row continues the modular design with a Treemap (`treemap_chart.py`) that visualizes the distribution of artefacts by Field of Science (FoS). Each block represents a FoS L3 category, scaled by the count of distinct artefacts, and supports interactivity via clickable regions or a reset button (“Clear FoS filter”). Adjacent to the treemap is a second Grouped Bar Chart (`grouped_barplot.py`) that highlights reproducibility by displaying RI and RCCI metrics across research output categories. To ensure consistency, all six visualizations dynamically respond to user interactions and filter selections, offering a seamless and informative exploration of the dataset. To maintain visual consistency, each of these graphs updates dynamically in response to filter selections and chart interactions.

The file `callbacks/overview_callbacks.py` defines the dynamic behavior of the Overview tab. A single callback function, registered via the `register_overview_callbacks()` function, handles all interactivity and output rendering. The callback listens to multiple inputs: dropdown filter values, chart clicks (e.g., Barplot, Treemap), and reset button clicks (e.g., Clear country or Clear FoS filters).

In order to provide synchronized changes across all dashboard components, the callback in the Overview tab adheres to a defined and modular logic. In order to enable time-based comparisons, the dataset is first filtered using the utility function `filter_data_by_selections()` from `utils/data_filter.py`, which pulls data for both the chosen year and the prior year. The algorithm conditionally adds additional filtering to refine the view based on the source of the interaction, such as whether a user clicked a country in the barplot or a field of science in the treemap. A reset button ensures usability and flexibility by returning the filtering state to its initial setup. Once filtering is complete, updated KPIs are generated using `generate_kpis()` and formatted through `generate_kpi_section()` for display. The filtered datasets are then sent to specific visualization methods that produce updated figures in real time, such as `create_pie_chart`, `create_barplot`, `create_grouped_barplot`, and `create_treemap_chart`. This flow guarantees that every tab element is responsive, consistent, and contextually correct.



The COLUMN_MAP from settings/column_config.py is used to refer to dataset columns during the callback. This schema decouples logic from specific column names, allowing seamless adaptation to future changes in the dataset structure.

The general layout adheres to the concepts of modularity and reuse. The graph_card() utility function in overview_layout.py is used to construct graph cards, standardizing the look and functionality of every visual component. Cards come with a dynamic Plotly graph, a consistent header, and optional buttons to govern interaction. The selection_boxes() function centralizes selection logic and filters, guaranteeing uniform tab-to-tab styling and behavior. Furthermore, without interfering with the user experience, the PDF guide button enhances user advice and documentation integration.

The Overview tab's structure balances visual aesthetics with performance and usability. The tab is extensible and maintainable thanks to its responsive layout, centralized filtering logic, and schema abstraction. Changes can be made separately without affecting the overall application framework, whether it's updating a metric, swapping out a graph type, or adjusting to a new dataset.

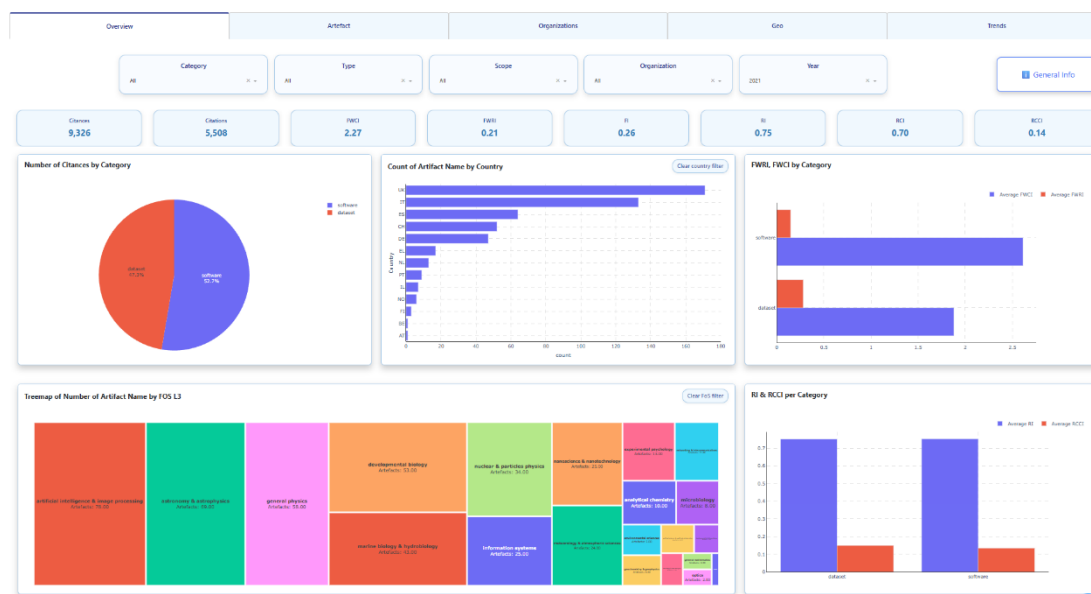


Figure 7: Overview Tab of Python Dashboard

4.2.3 Python Dashboard – Artefact Tab

The Python-based dashboard's Artefact tab is intended to provide an artefact-centric perspective of impact, reproducibility, and reuse metrics. By bringing to light measurements associated with individual artefacts, it broadens the analytical focus from broad trends to particular research outcomes. This makes it possible for users to determine which artefacts are often utilized, heavily referenced, or vehemently defended or contested in scientific discourse.

To preserve consistency and maintainability, the tab layout is defined in `layouts/artefact.py` using the same modular structure as the other tabs. Users can dynamically modify filters using dropdown selection boxes for Scope, Organization, Type, Category, and Year. Additionally, a prominently styled "General Info" button connects to PDF versions of supporting documentation.

The filtered dataset is used to update the KPI section at the top of the page, which offers a concise synopsis of the most important performance indicators pertaining to artefacts. A snapshot of artefact-level performance is provided by each KPI, which is updated to take into account the current filter state and includes signs of reproducibility and reuse.

There are six primary visual elements on the Artefact tab. The Top 5 Artefacts by RI chart is a horizontal barplot displaying the artefacts with the highest average Reusability Index values, categorized by type. In addition, the Metrics by Artefact table provides a detailed view of multiple indicators for each artefact, including RI, RCCI, and FAIR-related scores.

The second row of visualizations enables multidimensional exploration. The Top 5 Artefacts by RCCI is presented as a treemap, emphasizing artefacts with the highest composite reproducibility scores. A Top 10 Countries pie chart provides a geographic breakdown of artefact distribution. The third visualization, Supported & Refuted Artefacts by Field of Science (FoS), is a combo bar and line chart that compares the number of supporting and refuting

statements against the number of artefacts per field, offering insight into the evidential weight and controversy surrounding specific research areas.

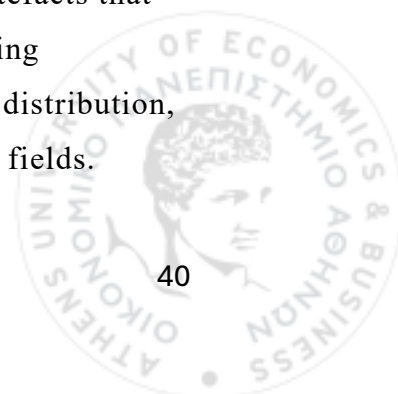
Each visualization is interactive. Clicking on an artefact, country, or FoS updates all visuals and KPIs accordingly. Each chart has clear filter buttons that let users go back to the default filtered view and reset their selections.

The callback logic which is used in the Artefact tab, is defined in `callbacks/artefact_callbacks.py`. A multi-output callback listens for changes to the dropdown filters and chart interactions. The `filter_data_by_selections()` function is used to apply base filtering, and the dataset is then conditionally narrowed further based on the user's chart selections (e.g., drilling down to a particular object or nation).

Clicking the "clear" button resets the corresponding filter and returns the entire filtered dataset. `Generate_kpis()` is used to regenerate KPI values, while `generate_kpi_section()` is used to format them for display. Then, using the corresponding functions from the `graphs/` subdirectory, all updated charts—barplot, pie chart, treemap, table, and combo chart—are displayed.

Special care is taken in the combo chart to handle long field names and to apply custom hover logic that maintains interpretability. To improve user experience without compromising detail, labels are trimmed for layout clarity while keeping their full names in tooltips.

The Artefact tab follows the same structural and visual design guidelines as the rest of the dashboard, which include a clean card-based structure, a gentle blue color scheme, and device-responsive formatting. Consistent user interaction patterns are made possible by the combination of each card's title, visualization, and optional filter-reset button. The Artefact tab supports effective reuse in many dashboard scenarios and maintains flexibility to changes in data format through the use of centralized column mappings and modular plotting algorithms. This tab highlights the particular artefacts that create scientific effect, reinforcing the dashboard's goal of fostering transparency and repeatability. It enables users to investigate the distribution, citation, and discussion of these artefacts in a variety of research fields.



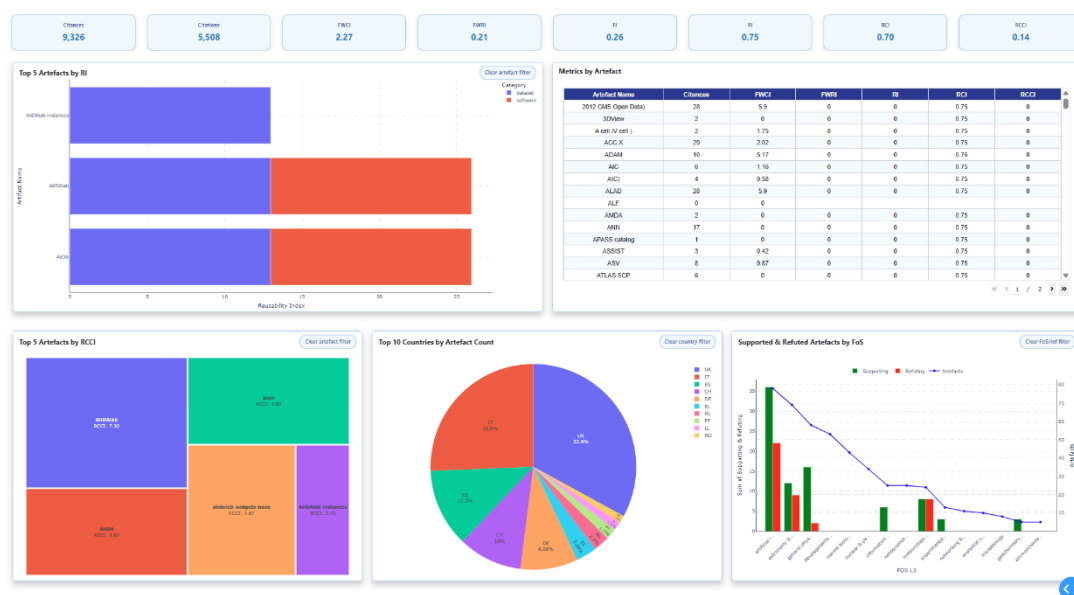


Figure 8: Artefact Tab of Python Dashboard

4.2.4 Python Dashboard – Organization Tab

The Organization tab gives an overview of the distribution of reproducibility & reusability metrics across different research organizations. It is intended to facilitate comparisons between institutions by providing aggregated performance indicators, with a focus on field-normalized results and sector contributions. This tab links entity-level accountability to thematic research themes.

The tab layout, defined in layouts/organization.py, follows the consistent modular design of the dashboard. On top of the interface are filter controls, being; Scope, Organisation, Type, Category, and Year, with an additional PDF information button for further documentation. The body of the tab is split into two sections of visualisation elements, and the KPI section is represented at the bottom of the page.

The first visualization row features a stacked bar chart showing the Top 10 organizations by number of artefacts, segmented by Field of Science (FoS). This enables users to understand not only organizational output volume but also the disciplinary distribution of contributions. The Metrics by

Organization table, which offers a detailed analysis of metrics including Reusability Index (RI), Reproducibility Composite Confidence Index (RCCI), and FAIR scores per organization, is located next to this chart.

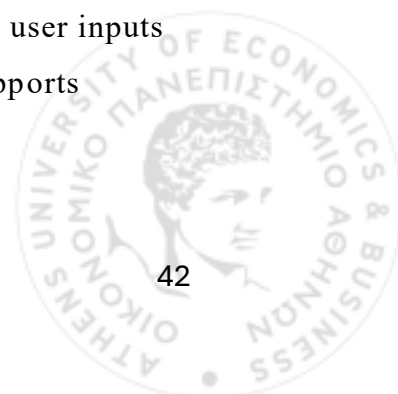
The second row includes two vertical barplots: the Top 5 Organizations per RI and the Top 5 Organizations per RCCI, providing a side-by-side comparison of performance in terms of reproducibility and reuse. Each chart is interactive, allowing the user to drill down by clicking on an organization bar or specific FoS segment.

The core logic for this tab is contained in the `register_org_callbacks()` function located in `callbacks/organization_callbacks.py`. The callback listens to both filter dropdowns and user interactions with the three charts: the stacked barplot and the two indicator-based barplots.

`Filter_data_by_selections()` is used to perform a base filtering step upon any user interaction or dropdown selection, isolating data for the selected year and other selected dimensions. Depending on the chart that is triggered, additional conditional filtering takes place. When a user clicks on the stacked bar chart, both the selected organization and its corresponding Field of Science are used to drill down into more specific subsets of the data. In contrast, interactions with the RI or RCCI charts apply a narrower filter that isolates the dataset based solely on the selected organization. To support exploratory flexibility, a “Clear org filter” button is also included. When clicked, this button resets the view, removing any chart-based filters and returning the dataset to its default state as defined by the current dropdown selections.

Following filtering, `generate_kpis()` is used to renew the KPIs, which are then supplied to `generate_kpi_section()` for display. The corresponding function is then used to construct each chart: `create_barplot()` for the RI and RCCI comparisons, and `create_stacked_bar_chart()` for the organizational breakdown. `create_metrics_table()` is used to construct the data table.

Each visualization is guaranteed to stay completely responsive to user inputs thanks to the modular and reusable callback logic, which also supports uniform performance across organizational views.



The Organization tab is essential for stakeholders seeking to evaluate and benchmark institutional contributions to reproducible science. The dashboard helps users to find leaders and gaps in institutional practices by highlighting top performers in RI and RCCI and segmenting outputs by discipline. By enabling users to go into particular companies and examine their metric composition in context, it also promotes accountability and transparency.

Similar to other tabs, the visual design has a simple, card-based look with well-balanced color schemes and user-friendly interaction. This guarantees that users may decipher intricate data relationships without experiencing cognitive overload, facilitating better decision-making and focused funding insights.

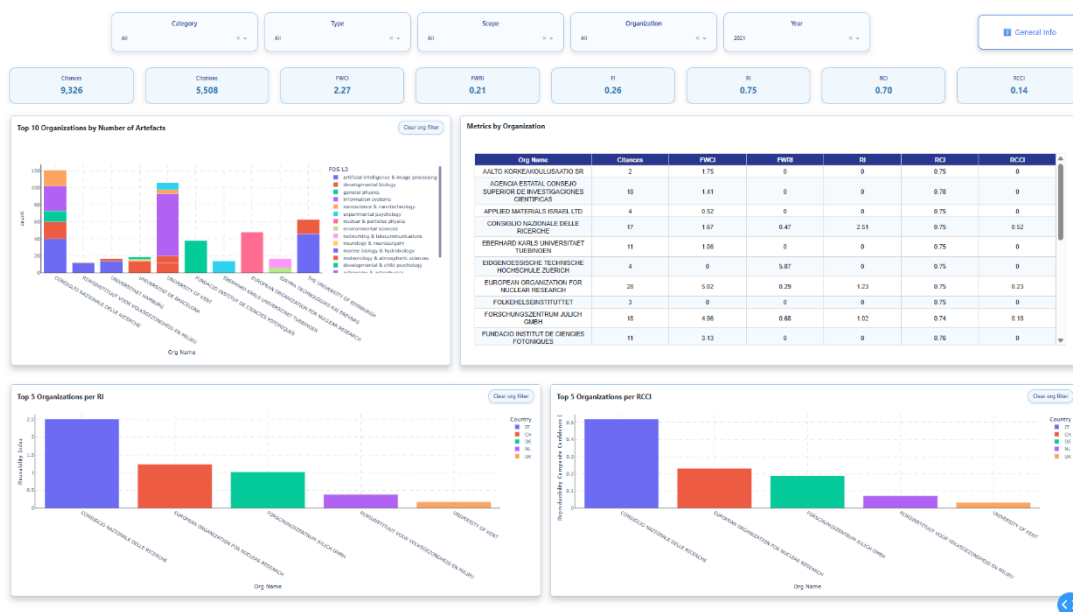
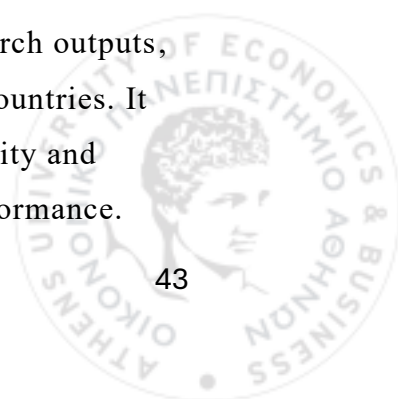


Figure 9: Organization Tab of Python Dashboard

4.2.6 Python Dashboard – Geographic Tab

The Geographic tab provides a geographical perspective on research outputs, highlighting how artefact-related metrics are distributed across countries. It enables stakeholders to explore geographic trends in reproducibility and reusability and identify regional patterns or gaps in research performance.



This tab serves both analytical and comparative purposes by combining map-based interactivity with detailed metric tables.

The layout for this tab is defined in `layouts/geographic_layout.py`, following the same modular and responsive architecture as other tabs. At the top, users can access dropdown filters for Scope, Organization, Type, Category, and Year. A prominently placed PDF info button links to supporting documentation, maintaining consistency with the design across the dashboard.

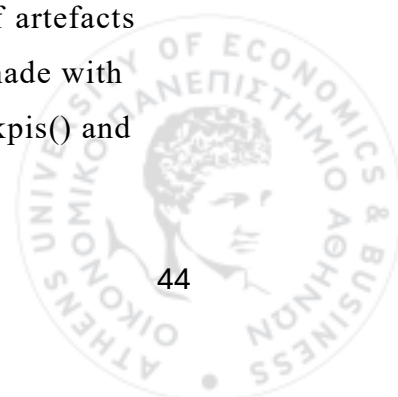
A dynamic KPI area beneath the filters offers a real-time summary of the overall performance for the chosen year. The choropleth map in the first row of visual material is titled "Number of Artefacts per Country." It uses ISO3 country codes to display artefact counts by nation and lets users click on a region to go further into that nation's data [33].

Metrics by Organization and Metrics by Artefact are displayed in two long, scrollable tables in the second row. These tables, which offer detailed, tabular insights into each entity's repeatability indicators and enable side-by-side comparisons across regional boundaries, are filled in according to the user's choices on the map or dropdown menus.

`Callbacks/geographic_callbacks.py` contains the main callback mechanism, where a single multi-output callback controls the tab's overall behavior. Along with click events on the choropleth map and a "Clear country filter" button, it listens for changes in the dropdown filter settings.

Filtering starts with a call to `filter_data_by_selections()`, which divides the dataset into current-year and filtered subsets based on the filters that were chosen. Using the `convert_iso2_to_iso3()` function, an iso3 country code is added to each dataframes for geographic reasons based on their country column. The depiction does not include any rows that do not have valid country codes.

The data from the current year is always used to create the choropleth map and KPI section. The map figure, which shows the distribution of artefacts and performance measures like RI and RCCI among nations, is made with `create_map_chart()`. The KPI cards are generated with `generate_kpis()` and presented with `generate_kpi_section()`.



Clicking on a nation on the map further filters the information used in the organization and artefact tables by using the iso3 value from the clicked location. The tables are reconstructed using the entire current-year dataset, eliminating any geographic drill-down filters, if the clear option is clicked. Updated KPI cards, the choropleth map, and two tables produced with `create_metrics_table()` utilizing the relevant entity columns are all included in the final result.

The Geographic tab enables an understanding of how reproducibility and reusability vary across regions. It supports comparative geographic analysis, helping identify national or regional leaders and laggards in terms of research quality and transparency. The tab facilitates multiscale insights, ranging from institutional or artefact-level observations to national patterns, by fusing map-based filtering with thorough metric breakdowns by organization and object.

This tab follows the main design tenets of the dashboard, which include adaptability across screen sizes, interactive modularity, and clean aesthetics. Both high-level summaries and in-depth analyses of country-specific indicators are supported by the flexible design, which guarantees that users may engage with geographic data in an easy and educational manner.

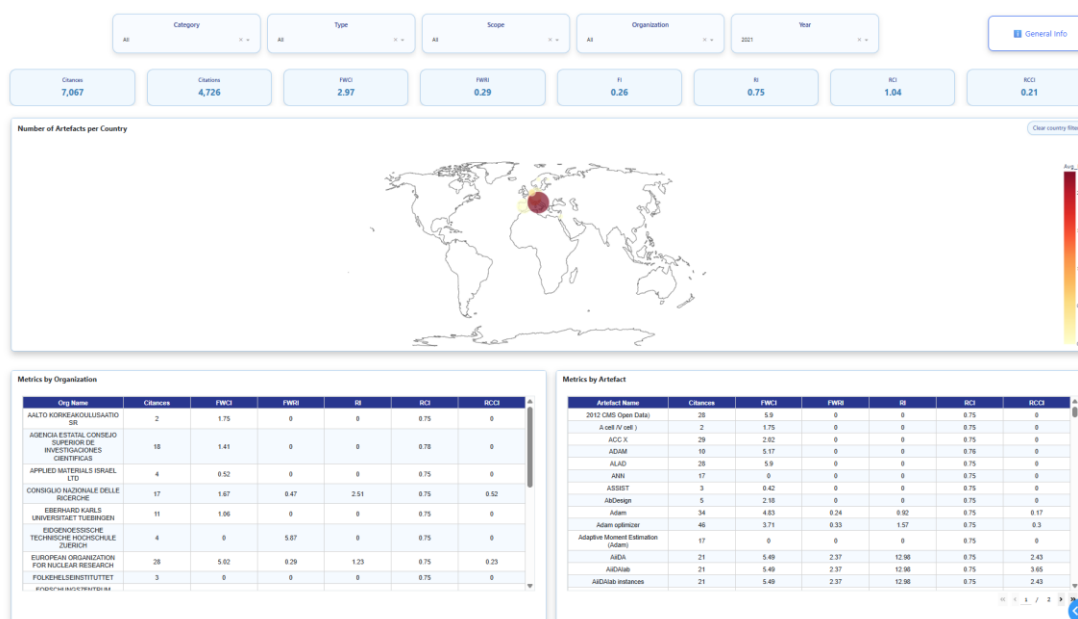


Figure 10: Geographic Tab of Python Dashboard

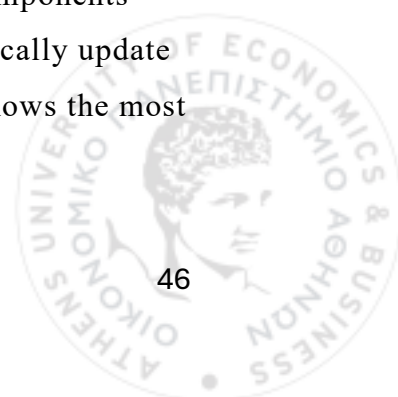
4.2.7 Python Dashboard – Trends Tab

A longitudinal view of research metrics is offered by the Trends tab, which enables users to track changes over time and determine if important indicators are rising or falling. When assessing the efficacy of programmatic financing, policy changes, or reproducibility initiatives across a number of years, this tab is very helpful. The tab facilitates both high-level trend analysis and metric-specific studies by combining dynamic time series charts and Year-over-Year (YoY) tables.

The layout is defined in `layouts/trends_layout.py` and adheres to the modular, filter-driven design approach used across the dashboard. Five selectable filters—Scope, Organization, Type, Category, and Year—as well as a stylized General Info button that connects to the dashboard's documentation are displayed to users at the top. The KPI section that follows shows summary statistics from the whole historical timeline of the dataset, not just the chosen year. A more comprehensive framework for analyzing performance trajectories is provided by this all-time image. Users can analyze changes in citances and related metrics across consecutive years, grouped by field, using the full-width table named "YoY Growth of Citances by Field of Science" which follows the KPI cards.

The second visual row includes three side-by-side line charts. The first tracks Supporting and Refuting Citances over time, providing insight into the sentiment and validation dynamics of citations. The second chart shows Artefact Overtime, highlighting the annual emergence of distinct artefacts across different categories. The third plot visualizes changes in Reusability and Reproducibility Scores (RI and RCCI) across years, making it easy to identify whether these quality indicators are improving.

All visual elements are built inside consistent, shadowed card components that adjust smoothly and are easy to read. Filter choices automatically update all KPIs, tables, and charts, making sure the dashboard always shows the most relevant and current information for the selected options.

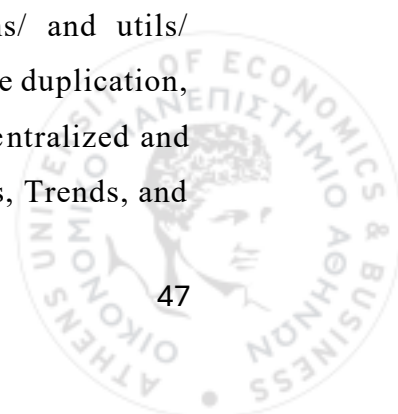


Callbacks/trends_callbacks.py contains the callback mechanism for this tab. It creates visuals from filtered data using a multi-step process. Since the time-based analysis necessitates access to the entire dataset, the callback starts by applying the chosen filters across all dimensions—Scope, Organization, Type, and Category—with the exception of the year filter. For this, the filter_data_by_selections() function is employed. Next, it uses generate_kpis() and generate_kpi_section() to create the KPI section from this filtered historical data. These KPIs offer a fixed point of reference for comprehending the more general patterns displayed in the visualizations.

Three line charts are then constructed. The Supporting & Refuting Citances chart uses grouped totals per year to plot comparative citation sentiments. The Artefact Overtime chart tracks the count of unique artefacts by year and category, revealing peaks or declines in research production. The final chart, RI & RCCI Overtime, plots yearly averages of reusability and reproducibility metrics, allowing users to assess whether these scores are progressing.

The YoY Growth table is generated using generate_yoy_table(), which compares metrics between the selected year and the previous one. These include Citances, Citations, Reproducibility Confidence Index (RCI), and Reproducibility Composite Index (RCCI). The resulting table is styled using Dash's dash_table.DataTable, with conditional formatting and standardized column naming for clarity (e.g., "YoY RCCI %"). These visual elements help users understand year-specific findings in the context of broader time trends, making it easier to see if interventions and policies are having a real impact over time.

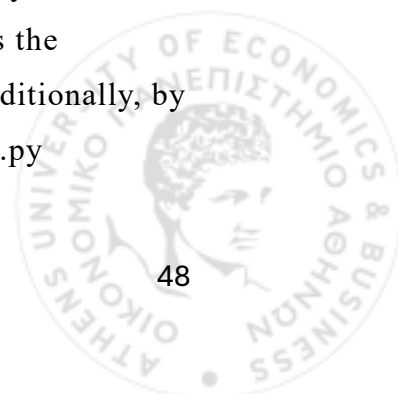
Every visual component—whether a table, map, or chart—has been created as a reusable element in order to smoothly support the functionality explained across all dashboard tabs. Each tab may dynamically produce insights while preserving a consistent visual style thanks to the abstraction of these components into modular utility functions kept in the graphs/ and utils/ directories. This method improves maintainability, minimizes code duplication, and guarantees design coherence. The fundamental logic stays centralized and flexible as users move between tabs like Overview, Organizations, Trends, and



Geography, enabling small parameter adjustments to reuse the same visualization capabilities for other analytical settings. The dashboard's scalability is greatly influenced by its modular design, particularly in situations when the data format or analytical focus changes. To ensure modularity, code reuse, and consistent dashboard appearance, all charts and tables were abstracted into reusable utility functions located in the `graphs/` and `utils/` directories. This architecture reduces logic duplication and allows insights to be dynamically rendered on each dashboard tab.

Every visualization was created as a reusable component inside the `graphs/` directory to provide uniformity, maintainability, and modularity throughout the dashboard. Flexible parameters enable each visualization function to adjust to various data settings without requiring code duplication. To ensure visual coherence throughout the application, all charts are styled according to a single visual concept that includes white backgrounds, light grey dashed gridlines, and clear font. Simple yet effective charts like pie charts, line plots, and bar charts are among the fundamental visualization components. With choices for aggregate and color encoding, the `barplot.py` package makes it possible to create basic horizontal or vertical bar charts. Likewise, dynamic line plots that provide category-based differentiation and temporal trends are provided by `line_chart.py`. In order to display categorical distributions as proportional slices with percentage labels embedded within the chart, the `pie_chart.py` module computes either raw counts or aggregated data.

Using combination charts and grouped visual comparisons, more intricate analytical visualizations are produced. The `combo_chart.py` module is perfect for displaying composite trends, such the quantity of artefacts versus citation metrics across entities, because it creates dual-axis combo charts that combine two bar metrics with an overlay line. A program called `stacked_bar_chart.py` stacks metric values or counts within grouped categories to display both internal and total distributions, while `grouped_barplot.py` enables the comparison of various metrics side-by-side across categories. Additionally, by utilizing nested fields like Field of Science levels, `treemap_chart.py`



facilitates hierarchical examination of data and offers an easy-to-understand breakdown of volume or impact by category.

In terms of spatial analysis, map_chart.py produces interactive choropleth-like maps using scatter geo-plots. Each marker's size and color represent metric intensity and reproducibility indicators for each nation. A worldwide overview of reproducibility and reuse activities is made possible by its automatic conversion of ISO codes and integration of average metric computations.

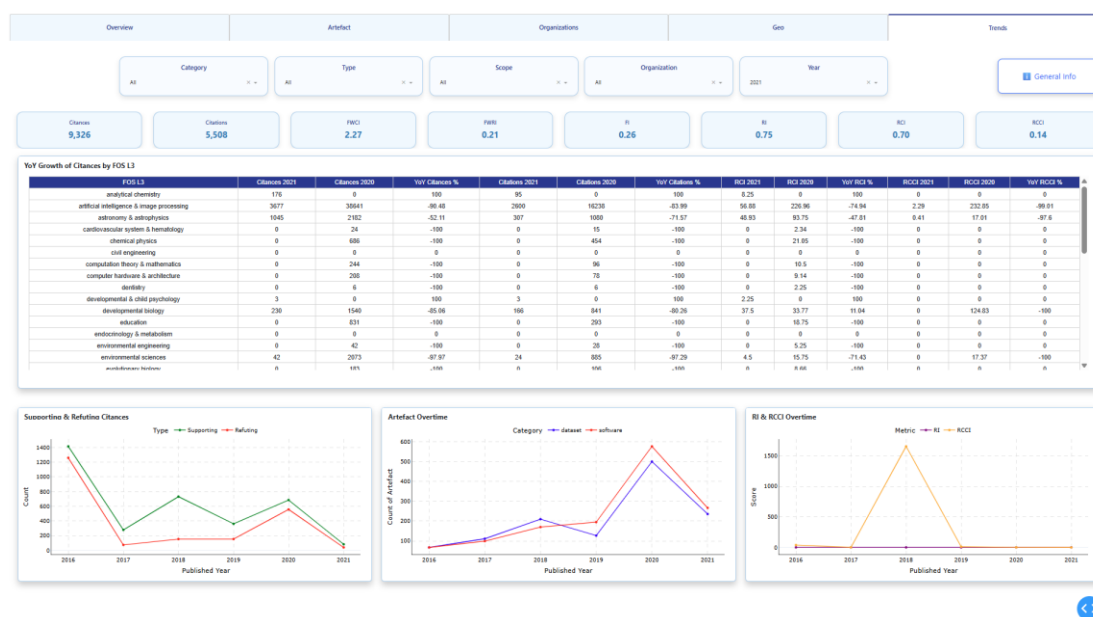
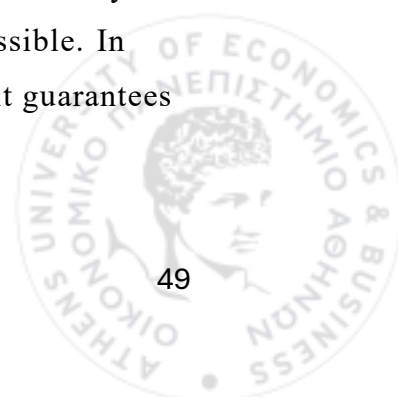


Figure 11: Trends Tab of Python Dashboard

Table.py dynamically creates interactive Dash DataTables that summarize key performance metrics at the organizational or artefact level for tabular representations. The most pertinent entries are highlighted in these tables using conditional formatting and alternate row shading, which also feature average metric computations and a total summary row. Additionally, by computing year-over-year variations for a number of metrics organized by specific attributes, yoy_table.py makes temporal comparisons possible. In addition to clearly displaying growth patterns or dips over time, it guarantees reliable handling of zero values and missing data.



Together, these reusable graph modules provide the foundation of the visual analytics layer of the dashboard. They permit the platform's scalability and future extension to accommodate additional indicators or analytical parameters in addition to fostering clarity and standardization.



5. Comparison & Evaluation

To objectively assess the strengths and limitations of each dashboarding approach, a comparative evaluation was conducted based on the criteria defined in the Methodology. Python (Dash) and Power BI both effectively reproduced the intended features, however they differ greatly in terms of long-term maintainability, flexibility, development effort, and visual refinement. The organized comparison in this section is followed by a thorough examination of the trade-offs and factors to be taken into account when choosing one tool over another for real-world applications.

5.1 Comparison Table

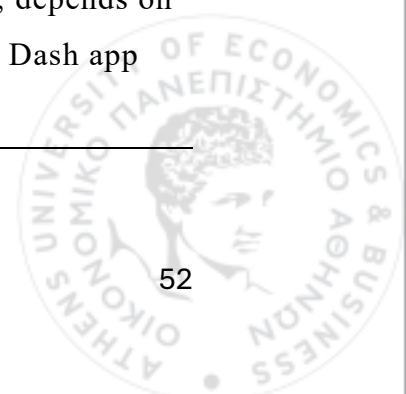
Visual quality, interactivity, customization, development effort, performance, reproducibility, flexibility to schema changes, scalability, and stakeholder communication are among the evaluation criteria that were established in the Methodology. The two parallel implementations created in the earlier chapters—the Power BI dashboard and the Python-based dashboard created with Dash and Visual Studio Code—were evaluated using these dimensions. The Python dashboard placed more emphasis on code-driven flexibility, modularity, and reusability across datasets with comparable structure than the Power BI version, which focused on accessibility, quick prototyping, and refined visuals utilizing built-in components.



The following table presents a structured, side-by-side comparison of both tools across the selected evaluation dimensions:

Table 1: Technical and Functional Comparison of Power BI and Python Dash

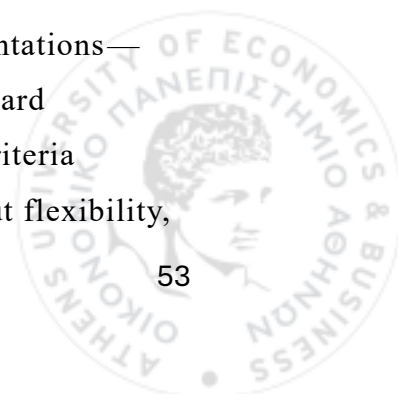
Criteria	Power BI	Python (Dash)
Visual Quality	High visual polish using native cards, charts, and map components with minimal design effort. Unified aesthetics and consistent layout by default.	Clean and minimal styling achieved through custom layout and Plotly white theme; requires manual effort for spacing, shadows, hover info, and color harmonization.
Interactivity	Built-in slicers, cross-filtering, drill-through buttons, and map tooltips; consistent across all tabs.	Manually implemented callbacks using Dash; supports dynamic filtering, custom toggles (e.g., Scope), interactive hover info, and export features.
Customization	Limited to built-in visuals and calculated DAX logic; some layout or control options restricted.	Full control over layout, logic, and behavior. Custom components (e.g., Scope toggle, help/export buttons) integrated through code.
Development Time/Effort	Fast prototyping with low-code environment; ideal for analysts without programming skills.	Requires setup in VS Code with Python and Dash; higher development effort but more adaptable in the long term.
Performance (Loading, Filtering)	Highly optimized backend with instant filtering and smooth tab switching.	Good performance for moderate datasets; depends on filtering logic and Dash app optimization.



Criteria	Power BI	Python (Dash)
Reproducibility	Low reproducibility; limited version control and transparency of logic; harder to trace/filter dependencies.	High reproducibility via modular codebase; Git versioning, config-driven filtering, and centralized abstraction improve traceability and collaboration.
Adaptability to Schema Changes	Manual updates needed when column names change; visuals are tightly coupled to original structure.	Highly flexible through COLUMN_MAP; dynamic column referencing enables seamless adaptation to renamed or restructured datasets.
Scalability & Maintainability	Best for single-use or medium-scale deployments; larger dashboards may become complex to manage without standard code reuse.	Designed for scalability: modular callbacks, reusable graphs, and separation of logic/layout support long-term extension and multiple datasets.
Stakeholder Communication	Extremely accessible; ideal for non-technical users and business reporting. Easy export to PDF or web publish.	Requires web deployment (e.g., Dash Enterprise or Heroku); better for technical stakeholders needing transparency and integration with scientific workflows.

5.1.1 Visual Comparison Snapshots

This section offers visual comparisons between the two implementations—Power BI and Python Dash—highlighting how equivalent dashboard components look and act in each tool to support the evaluation criteria previously stated. The trade-offs between visual aesthetics, layout flexibility,



interactivity, and development work are clearly illustrated by these side-by-side examples. Dash offers fine-grained flexibility but necessitates manual setup, whereas Power BI prioritizes simplicity of use and refined visuals through built-in components.

KPI Cards

Power BI's Overview tab uses native card visuals to display key performance indicators with minimal configuration. These visuals are styled automatically, ensuring uniform alignment and consistent formatting.



Figure 12: Power BI KPI cards with refined layout

By contrast, the Dash version constructs KPI cards using reusable HTML Divs and Bootstrap layout classes, with metrics passed dynamically through callbacks. This approach offers more control over layout and spacing but demands explicit styling.



Figure 13: Dash KPI cards with dynamic updates

Stacked Bar Chart

In the Organizations tab, Power BI enables users to quickly build a stacked bar chart grouped by Field of Science (FoS) using drag-and-drop features. The visual includes category-based color coding, legends, and auto-sized axes.

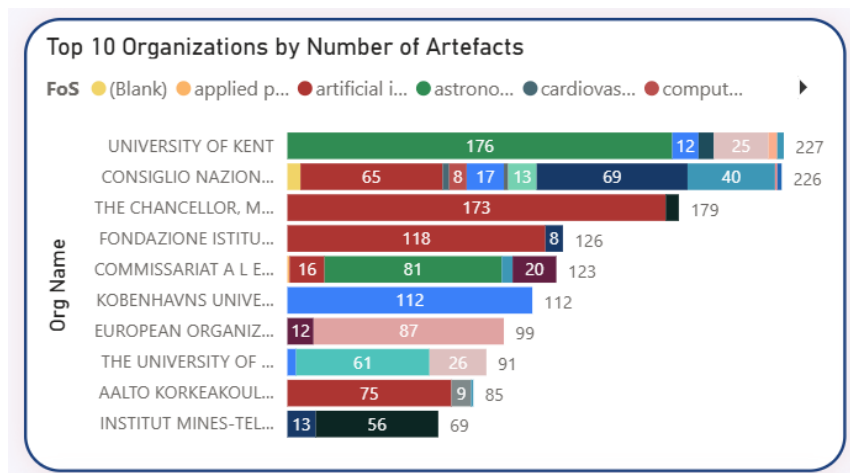


Figure 14: Power BI stacked bar chart of top 10 organizations by artefacts

In Dash, the same chart is implemented using Plotly Express, where colors, layout, and interactivity must be manually configured. Although this requires more code, it provides full control over formatting and export behaviour.

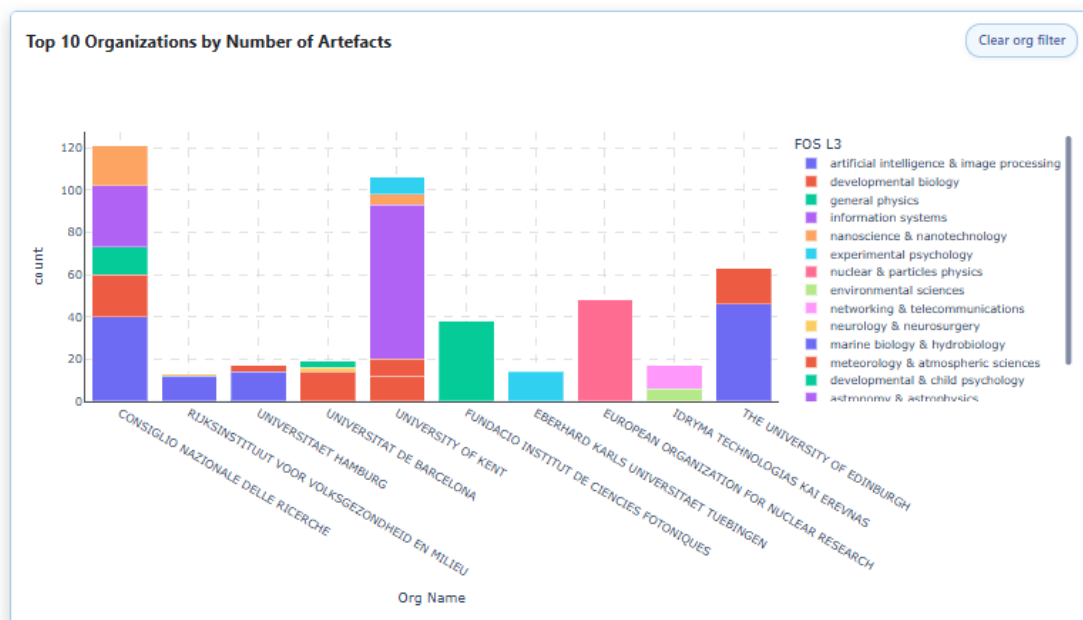


Figure 15: Top 10 Organizations by Number of Artefacts



Line Charts

The Trends tab features three line charts in both dashboards. In Power BI, these are built using native visuals with consistent legends and hover tooltips. In Dash, the charts are generated using Plotly Express with custom logic for line color, mode, and layout. For example, the Artefact Overtime chart uses blue and red lines to visually differentiate polar values, enhancing interpretability.

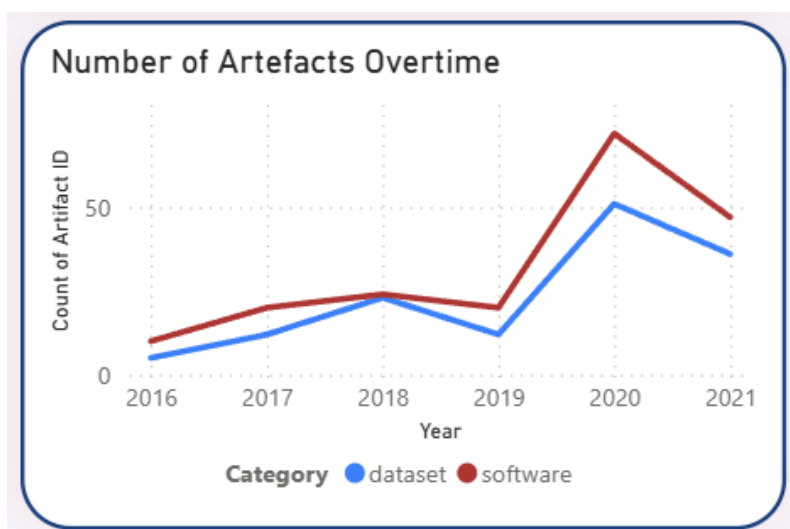


Figure 16: Number of Artefacts Over Time

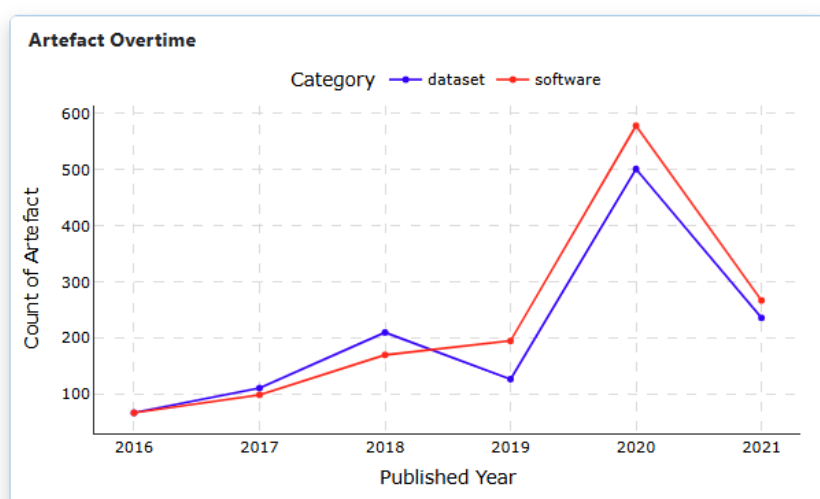


Figure 17: Artefact Publication Trends by Category

Year-over-Year Table

Power BI leverages its Matrix visual for a clean year-over-year (YoY) comparison table with built-in conditional formatting. The design is responsive and consistent with the overall theme.

Year	2019									
FoS	Citances	YoY Citances %	Citations	YoY Citations %	RI	YoY RI %	RCI	YoY RCI %	RCCI	YoY RCCI %
anesthesiology	68		8		0.00		0.75		0.00	
artificial intelligence & image processing	68	-99.62%	43	-99.58%	0.00	-100.00%	0.73	-2.76%	0.00	-100.00%
developmental biology	1709	-96.71%	863	-97.03%	544.33	-97.07%	0.76	0.79%	102.51	-97.99%
experimental psychology	236	807.69%	56	194.74%	85.13		0.75	-1.76%	15.80	
general chemistry	44		19		0.00		0.75		0.00	
general physics	374	-64.31%	155	-74.21%	0.11	-99.26%	0.75	-0.12%	0.02	-99.26%
geological & geomatics engineering	10		6		0.00		0.75		0.00	
information systems	14	-99.19%	9	-99.02%	0.00	-100.00%	0.75	-0.47%	0.00	-100.00%
meteorology & atmospheric sciences	4		2		0.00		0.75		0.00	

Figure 18: Year-over-Year Metrics by Field of Science

The Dash version uses a dash_table.DataTable with metrics dynamically calculated and styled. While it requires extra formatting logic (e.g., column renaming, rounding), it enables full customization of columns and values.

FOS L3	Citances 2019	Citances 2018	YoY Citances %	Citations 2019	Citations 2018	YoY Citations %	RCI 2019	RCI 2018	YoY RCI %	RCCI 2019	RCCI 2018	YoY RCCI %
anesthesiology	68	0	100	8	0	100	3	0	100	0	0	0
artificial intelligence & image processing	68	686	-90.09	43	288	-85.07	0.73	3	-75.71	0	5.35	-100
astronomy & astrophysics	0	138	-100	0	78	-100	0	2.25	-100	0	0.22	-100
chemical physics	0	126	-100	0	65	-100	0	0.74	-100	0	35.35	-100
computer hardware & architecture	0	24	-100	0	11	-100	0	0.75	-100	0	0	0
developmental biology	1709	1591	7.42	863	767	12.52	3.79	12.79	-70.35	512.55	29.66	1627.96
experimental psychology	236	0	100	56	0	100	1.49	0	100	31.61	0	100
general chemistry	44	0	100	19	0	100	0.75	0	100	0	0	0
general physics	374	61	513.11	155	36	338.56	2.25	0.75	199.81	0.66	0	100
geological & geomatics engineering	10	0	100	6	0	100	0.75	0	100	0	0	0
information systems	14	0	100	9	0	100	0.75	0	100	0	0	0
meteorology & atmospheric sciences	4	0	100	2	0	100	0.75	0	100	0	0	0
microbiology	70	52	34.62	30	27	11.11	1.5	1.51	-0.72	0	0.22	-100
nanoscience & nanotechnology	0	62	-100	0	40	-100	0	1.52	-100	0	2.61	-100
networking & telecommunications	0	121	-100	0	76	-100	0	2.25	-100	0	0.45	-100
neuroscience & neuroanatomy	241	364	-33.76	98	102	-58.05	6.26	7.77	-31.26	3.71	7.71	17.14

Figure 19: YoY Growth of Citances and Reproducibility Metrics by Field of Science



Filters and Interactive Controls

Power BI offers built-in slicers, dropdowns, and drill-through actions that work seamlessly across tabs, enabling cross-filtering without extra setup. In contrast, Dash requires all interactive behaviors to be manually defined via callbacks. While this adds complexity, it allows for precise, context-aware features. For example, the "Clear artefact filter" button (shown below) resets selections and updates visuals accordingly—a level of control not natively available in Power BI. Dash's flexibility supports advanced interactions like cascading filters and toggle-based visibility, all handled through Python logic.

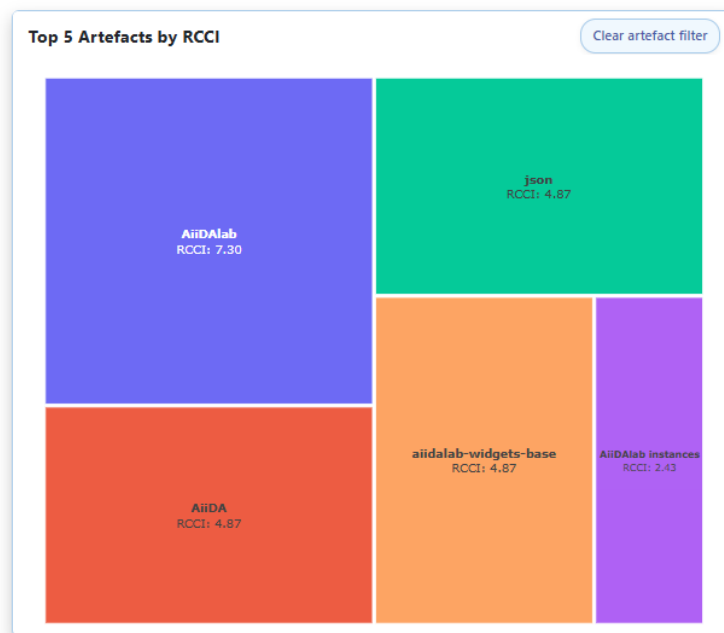


Figure 20: Top 5 Artefacts by Reproducibility Composite Confidence Index (RCCI)

Geographic Maps

Power BI provides rich geographic map visuals with automatic geo-detection, interactive tooltips, and smooth zooming.

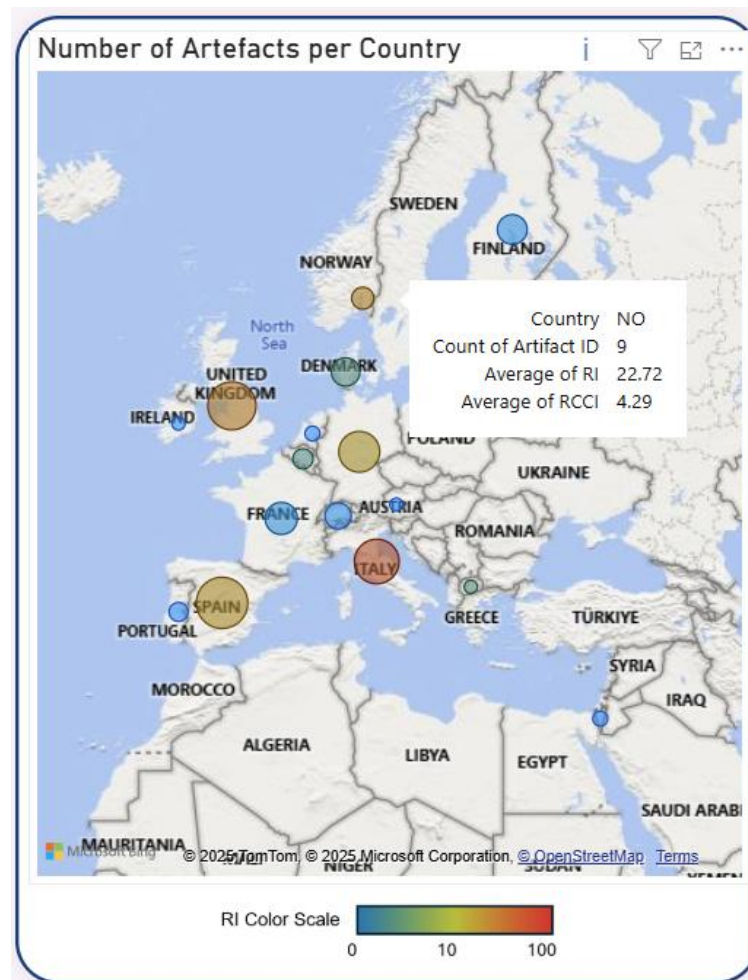


Figure 21: Geographic Distribution of Artefacts Across Europe

Dash supports geospatial visualizations such as scatter maps and choropleth maps through Plotly's mapping capabilities. While Power BI provides default map visuals with built-in country recognition and filtering, Dash requires explicit handling of geospatial data—either by providing country coordinates or integrating external GeoJSON files. This added complexity allows greater control over map aesthetics, interactivity, and logic.

In the example below, a scatter map was created to visualize the number of artifacts per country, colored by the average Reproducibility Index (Avg_RI). The tooltip displays contextual metrics (e.g., RCCI and artifact count), and a custom "Clear country filter" button resets the selected region, allowing users to explore other countries interactively. This level of control is useful in research dashboards requiring detailed filtering and dynamic metric exploration.

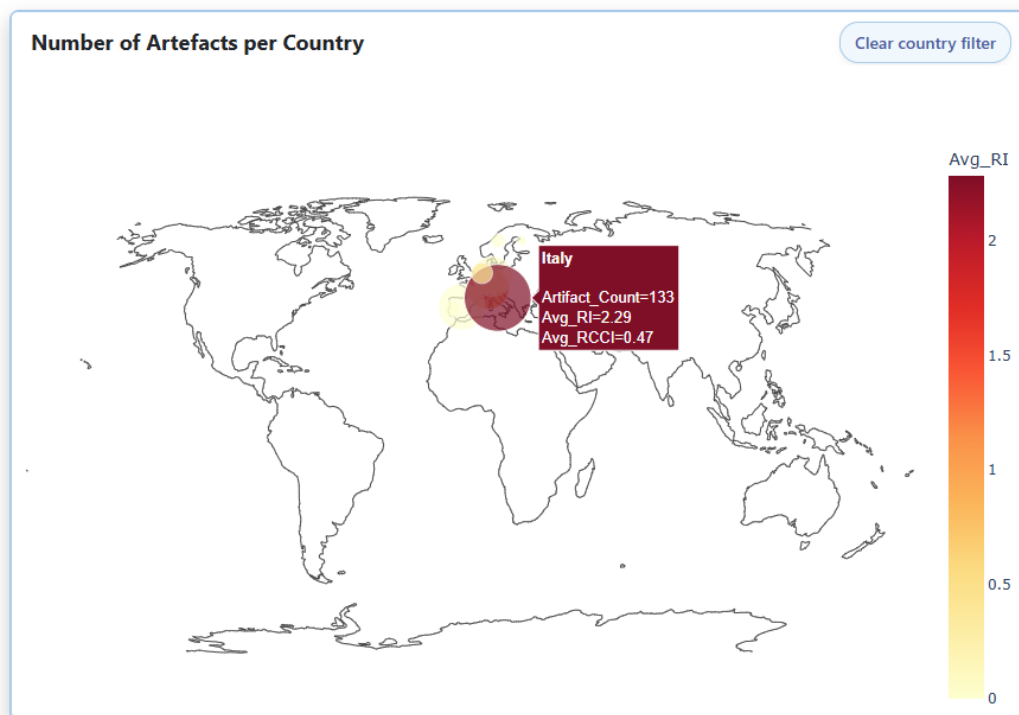


Figure 22: Global View of Artefact Distribution by Country

Tab Layout and Navigation

In Power BI, tabs appear as separate pages that are navigated through the left panel or embedded navigation buttons.



Figure 23: Dashboard Navigation Tabs

Dash simulates tabs through a combination of navbar links and dynamic layout rendering. Though this adds coding complexity, it enables control over URL behavior and page transitions.



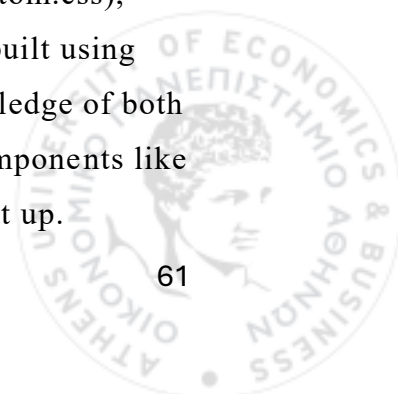
Figure 24: Section Navigation Tabs

5.2 Design and User Experience

Design and user experience were central to both dashboard implementations, though realized through fundamentally different design concepts.

Power BI is excellent at providing a well-designed, expert interface with minimal configuration. Its built-in visual components—such as cards, tables, treemaps, and maps—are drag-and-drop enabled, while its default color palettes, alignment tools, and responsive layout options make it easy to produce visually cohesive dashboards quickly. The platform is particularly useful for business stakeholders and external audiences because of its point-and-click usability, which even non-technical users may benefit from. In practice, the Power BI dashboard was more visually appealing and presentation-ready, particularly well-suited for embedding on the research center’s website. The out-of-the-box design quality allowed end users to interact with insights intuitively, without any technical setup.

On the other hand, despite requiring more work to design, the Python Dash implementation, which was created with Dash Core Components (dcc), Dash Bootstrap Components (dbc), and Plotly, provided a great deal of flexibility and control. Styles were defined in a custom CSS file (assets/custom.css), responsiveness had to be addressed explicitly, and layouts were built using Python code (dbc.Row, dbc.Col). This method necessitated knowledge of both programming and UI/UX concepts because even fundamental components like alignment, spacing, and hover behavior needed to be manually set up.



However, a more customized and reusable interface was made possible by this development effort. The Power BI version's design logic served as the inspiration for the dashboard's visual identity, which includes a white background, rounded KPI cards with shadows, and high contrast color schemes (such as red for citation metrics and blue for reproducibility). Custom Python callbacks (such as `overview_callbacks.py` and `geo_callbacks.py`) were used to power the interaction and provide fine-grained control over dynamic behaviours like filtering, tooltips, and export actions. Additionally, for uniformity between tabs, all design components (such as `kpi_section.py` and `selection_boxes.py`) were modularized.

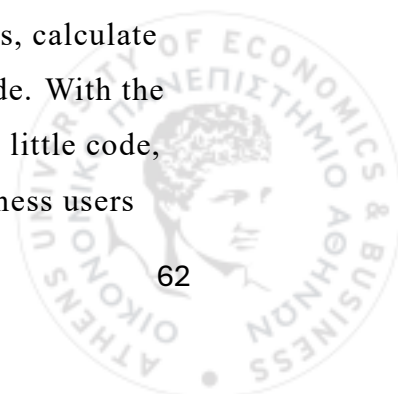
Despite being more time-consuming, Dash's adaptability was useful in technical and scientific contexts where reproducibility, version control, and customized processes are important. After it was set up, the Python dashboard offered a clear, reliable user experience that was ideal for use in a data science setting or internal research procedures.

In conclusion, Dash offers fine control, customisation, and design reusability, making it a solid choice for long-term research and engineering applications, while Power BI is perfect for quick, presentation-ready dashboarding with little effort.

5.3 Ease of Development

The development experience for Power BI and Python Dash diverges substantially in terms of required technical expertise, setup complexity, and long-term extensibility.

Using a drag-and-drop interface, prebuilt visuals, and integrated support for Power Query and DAX expressions, Power BI is made for quick dashboard creation. These tools enable users to perform data transformations, calculate custom metrics, and create polished dashboards with minimal code. With the help of these tools, users can construct beautiful dashboards with little code, compute custom metrics, and conduct data transformations. Business users

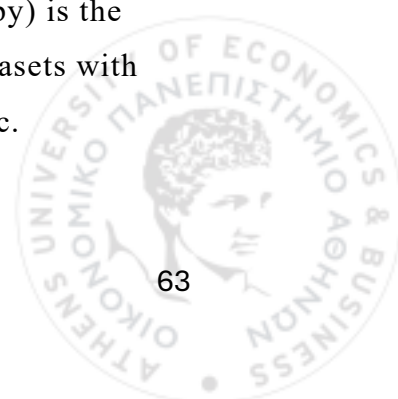


and analysts can easily create useful and aesthetically pleasing reports, especially if they are familiar with the Microsoft ecosystem. A visible, sequential approach to data reshaping is offered by Power Query, while complicated aggregations and calculated columns are made possible by DAX. Power BI is very accessible and efficient for quick prototyping and stakeholder involvement because of its low-code environment, which drastically cuts down on development time and the technical barrier.

However, Power BI has limitations when it comes to extensibility and reusability. While it supports moderately complex logic through DAX, advanced customization often requires workarounds or external tools. Because of its graphical interface-based capabilities, it is more difficult to programmatically reproduce visual logic across numerous reports or enforce code versioning. As a result, while initial creation is quick, it can be difficult to maintain complex interactivity or adapt the dashboard to substantially varied datasets.

In comparison, Python Dash requires a greater initial technical effort. Code must be used to explicitly describe each dashboard element, including layout, filters, interaction, and visuals. This dashboard, which was constructed using Visual Studio Code's Plotly Dash framework, had a modular design with well-defined subfolders for layouts, callbacks, graphs, and utility features. Dash's callback system, custom Python functions, and data wrangling using pandas were used to implement tasks like filtering, inter-tab synchronization, and KPI calculations. In addition, styling was done by hand using CSS in `assets/custom.css` or inline declarations.

This method has significant benefits in terms of reusability, flexibility, and integration with research procedures, despite requiring a strong grasp of Python, data manipulation, and fundamental front-end concepts. New metrics or tabs can be added with consistent logic once the modular code structure is established. Additionally, the configuration file (`column_config.py`) is the only thing that needs to be changed to adapt the dashboard to datasets with alternative column names without affecting the fundamental logic.



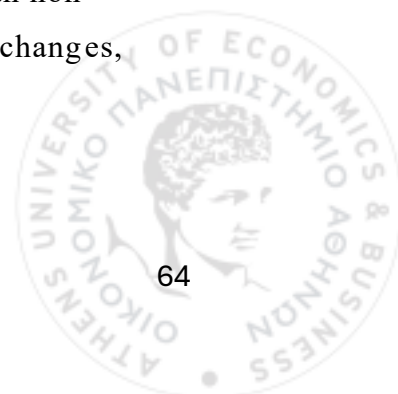
In this project, Power BI proved effective for fast, stakeholder-facing development, aligning visuals to institutional expectations with ease. It was very helpful for quick exploration and presentation. Dash was the superior option, though, for developing a dashboard template that is reusable and adaptable, one that can be maintained in version-controlled environments, support automation, and adjust to shifting structures. Therefore, Dash provides scalability, reproducibility, and complete control for technically complex use cases, while Power BI streamlines development for analysts.

5.4 Scalability and Maintainability

When selecting a dashboarding solution for long-term use, changing datasets, or multi-project deployment, scalability and maintainability are crucial. This dimension assesses each tool's ability to adjust to evolving complexity, data changes, and collaborative development requirements.

When it comes to data connectivity, Power BI provides instant scalability. With links to Excel, SQL databases, APIs, and cloud services, it easily interacts with big datasets. Because of this, it works incredibly well for rapidly visualizing large amounts of information with little setup.

Nevertheless, Power BI's scalability has drawbacks in terms of dashboard maintenance as data structures change. For example, it is frequently necessary to manually adjust Power Query steps, DAX measures, visualizations, and slicers when column names or formats in the source dataset change. Since these modifications cannot be handled programmatically, each instance needs to be updated separately. Additionally, Power BI files (.pbix) need extra technologies like Power Automate or SharePoint to provide versioning workflows because they are not version-controlled by default. As a result, while Power BI is excellent for rapid prototyping and sharing with non-technical users, it is less suited for environments where frequent changes, team collaboration, or code-based logic are expected.



Python Dash, on the other hand, was specifically created for modular scalability and maintainability in this project. Its architecture facilitates extensibility and clear organization by dividing logic, layout, and configuration into discrete subfolders. The COLUMN_MAP dictionary, which is kept in the column_config.py file, is one of its most potent features. Even when field names change, new datasets can be seamlessly integrated thanks to this abstraction layer, which separates interface functionality from the underlying dataset. To reflect these changes, developers simply need to update the mapping—not the core logic.

Dash's adherence to contemporary software development methodologies greatly strengthens the solution created for this project's maintainability. The dashboard facilitates scalable feature extension, structured change management, and collaborative development through the use of Visual Studio Code, Git-based version control, and a modular Python architecture. Reusable scripts and utility functions (filter_data_by_selections, create_barplot, kpi_section.py) encapsulate each functional piece, including filters, KPIs, and graphs, enabling centralized updates without changing the overall application logic. Despite requiring a more substantial initial development effort than Power BI, Dash produced a solid, reusable framework that could be tailored to changing datasets, research priorities, or institutional operations. The dashboard may be readily expanded to other projects or data domains with no effort thanks to the separation of callbacks and layouts and the use of COLUMN_MAP for column abstraction.

In conclusion, Power BI is still a good choice for static dashboards and quick reporting, particularly in settings with stable schemas and substantial data quantities. But in research contexts where version control, schema evolution, and workflow integration are essential for sustainability and reuse, the Python Dash implementation offers better long-term scalability and maintainability.



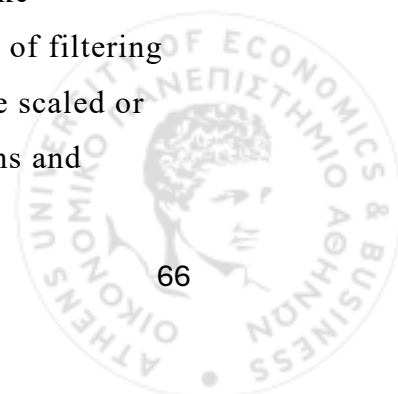
5.5 Code Reusability

Code reusability becomes crucial when dashboards need to be modified for new datasets, domains, or organizations. Although they do it in very different ways, Power BI and Python (Dash) both provide strong reusability methods. In actuality, they are equally reusable, with the best option based on technical preferences and the context of use.

With Power BI, users may duplicate layouts, themes, and logic structures across reports by using visual templates (PBIX or PBIT files). When the new datasets substantially resemble the old schema, it is efficient to reuse slicers, charts, DAX measures, and Power Query steps. Users can manually rebind visualizations or adjust transformations in Power Query's formula view and graphical interface, even if field names or data logic change. Power BI templates provide a useful and effective way to reuse data in settings where the schema is somewhat stable or where technical users can handle sporadic modifications. Additionally, the platform's visual-first approach speeds up new user onboarding and makes it easier for teams to reuse templates.

By contrast, the Python dashboard developed with Plotly Dash was built with reusability in mind from the start. It adopts a modular, library-style architecture, with layout, callbacks, graphs, and utilities housed in dedicated subfolders. Functions like `create_barplot`, `create_line_chart`, and `create_metrics_table` are fully generalized and parameterized for flexible reuse across tabs or projects. A central feature supporting reusability is the `COLUMN_MAP` dictionary, which abstracts column references, allowing the codebase to remain functional even when the input dataset changes. Instead of updating multiple scripts, users can adjust a single configuration file to accommodate new schemas.

Furthermore, by minimizing code duplication, utility functions like `filter_data_by_selections()` allow for the centralized management of filtering logic across all tabs. With this architecture, the dashboard may be scaled or modified with little alteration, primarily through layout extensions and configuration updates instead of rewriting the fundamental logic.

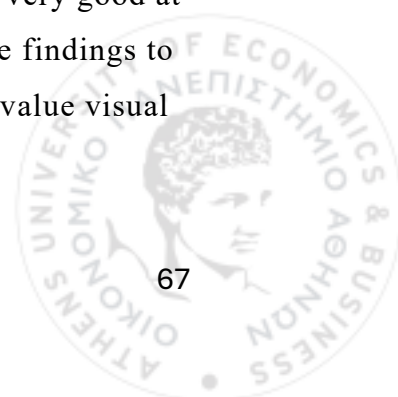


In conclusion, both Python Dash and Power BI may support code and dashboard reusability, although in different ways. With templates and interface-driven procedures that are simpler for business users to implement, Power BI excels in visual reusability. Python Dash, on the other hand, provides programmatic reusability through parameterized components, configuration files, and modular code. Both strategies are valid and successful; the decision is based on the team's technical proficiency and if logic abstraction or visual consistency is required.

5.6 User Experience and Stakeholder Communication

Effective stakeholder communication and user experience (UX) are essential components of any dashboarding system, particularly when the output is used to support funding allocation, guide strategic decisions, or convey research findings to non-technical audiences. Power BI has a clear edge in this area because of its well-designed user interface, simple navigation, and built-in interactivity capabilities.

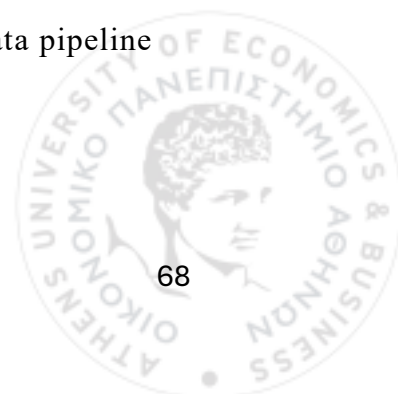
Without the need for programming, users can easily put together neat and coherent dashboards with Power BI's drag-and-drop canvas, uniform formatting, and extensive selection of prebuilt graphics. An overall feeling of visual professionalism is enhanced by the automatic harmonization of visual elements like slicers, cards, donut charts, and treemaps with pre-existing themes and layout guidelines. Tooltips, cross-filtering, drill-through actions, and bookmarks all contribute to increased interactivity by enabling users to explore the data organically and without the need for technical assistance. Power BI was implemented in this project using Power BI Service, which allowed for smooth integration with the research center's website and safe access via online portals. Because of these qualities, Power BI is very good at communicating with stakeholders. It may be used to communicate findings to funding agencies, research administrators, or policy makers who value visual clarity and usability.



The Python-based dashboard, which was created using Dash and Plotly, provides more granular customization and logic control, but it requires a more technical investment to design and use. Unless Dash Bootstrap Components or custom CSS are used, responsive layout behavior is not handled automatically, and each visual and interaction needs to be properly written. Greater design flexibility and analytical specificity are made possible by this, but the dashboard lacks Power BI's innovative visual refinement. Additionally, Dash needs a deployment pipeline, which usually entails internal servers, Heroku, or Dash Enterprise. Particularly for non-technical consumers who anticipate instant access and little setup, these processes create friction.

But when it comes to situations requiring extensive customisation, scientific reproducibility, or interaction with back-end data processing, the Python dashboard shines. It facilitates sophisticated programming logic that is more difficult to duplicate in Power BI, such as dynamic computations, metric changes, or algorithmic analysis. The Python dashboard in this project was made possible by features like modular callbacks, COLUMN_MAP-based filtering, and a reusable component architecture, which allowed researchers and data analysts to perform longitudinal study or gain deeper insights. Furthermore, it enables close interface with Python notebooks, data pipelines, or APIs, making multi-source aggregation or continuous data refreshes possible.

In conclusion, Power BI's sophisticated design, user-friendliness, and straightforward interface—which doesn't require any programming knowledge—make it ideal for dashboards that are visible to the outside world and high-level stakeholder communication. For decision-makers and larger audiences, it offers a simplified experience. However, the Python (Dash) solution is superior in terms of customization, analytical depth, and long-term reproducibility, although requiring a larger technical commitment. It is especially useful in internal or research-driven contexts where data pipeline integration, flexibility, and logic transparency are critical.

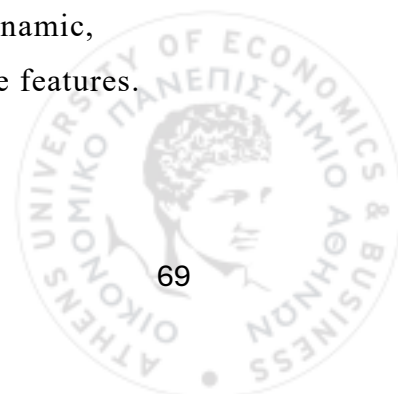


5.7 Reusability and template development

Reusability and template development are essential for maintaining and scaling dashboard solutions over time, particularly in research, evaluation, and policy environments where datasets are frequently updated or extended with new variables. This section evaluates how each dashboarding tool—Power BI and Python Dash—supports modular, repeatable, and adaptable dashboard development.

The Python dashboard was specifically made to be reusable. It was created with Dash and organized in a modular library-style architecture, or "lib format." Its design encourages maintainability, version control, and code generalization. This flexibility is supported by three essential characteristics. The same codebase can continue to function even if new datasets have renamed or restructured fields because of the centralized `COLUMN_MAP` configuration (in `column_config.py`), which separates dataset column names from the core logic. Second, teams may plug and play dashboard elements with little repetition because of the division of responsibilities among layout files, callbacks, utility functions, and reusable graph components (such as `create_barplot`, `create_line_chart`, and `create_radar_chart`). Third, all tabs filter data in the same way depending on user input thanks to a consistent utility layer (`filter_data_by_selections`), which lowers errors and improves maintainability.

Because only small configuration changes are needed, this modular and abstracted architecture not only facilitates expansion between tabs within the present dashboard but also permits replication across completely other datasets or institutions. Additionally, Python's ecosystem facilitates deployment flexibility (e.g., APIs, Jupyter Notebooks, Dash Enterprise), scheduled automation (e.g., cron or GitHub Actions), and connection with data pipelines. It is perfect for template-based deployments in dynamic, research-intensive, or analytics-focused contexts because of these features.



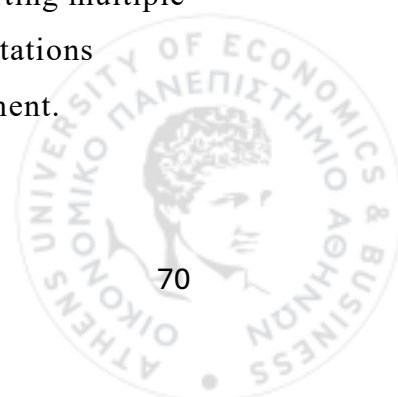
On the other hand, Power BI facilitates reusability mainly through its PBIT template files, which maintain layout and visual design. These templates can be very useful for stable datasets with constant schemas, enabling quick report replication while upholding visual standards. Nevertheless, the original data model is closely linked to these templates. Even minor modifications, like changing the name of a column, necessitate manual revisions in DAX measures, Power Query steps, and visual bindings. Reusability across evolving datasets is limited by Power BI's lack of support for centralized abstraction layers and dynamic schema adaption, in contrast to Python.

Power BI does not have integrated modular scripting or version control. Although Power Automate and similar technologies can provide some automation, they are not included into the main development process. As a result, maintaining templates across datasets or institutions becomes more manual and prone to mistakes. For non-technical users or settings where schema stability and visual consistency are the major objectives, Power BI templates are still useful in spite of this.

In contrast, Python Dash provides a more potent and adaptable method for templating and reusability. Programmatic logic, centralized column mapping, and modular code allow for the building of scalable, repeatable dashboards with no overhead. In contrast, Power BI is better at handling dynamic or research-intensive scenarios where flexibility, automation, and schema independence are essential. It also excels at template duplication and ease of use in static applications.

5.8 Performance and Loading Efficiency

Performance is a critical consideration in dashboard development, particularly when handling large datasets, applying dynamic filters, or supporting multiple users. Both Power BI and Python Dash exhibit strengths and limitations depending on the nature of the data and the deployment environment.



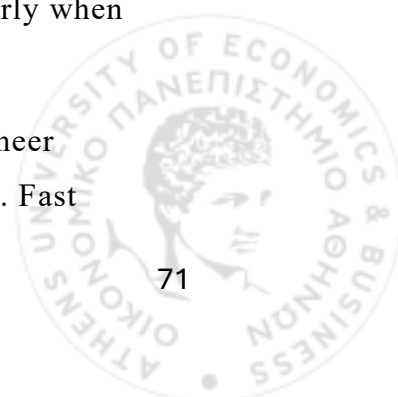
VertiPaq is a high-speed, in-memory engine that benefits Power BI by optimizing query performance and data compression in the background. Interactions like filtering, slicing, and cross-highlighting are managed quickly and effectively after the dataset is entered into the Power BI model. This eliminates the need for the user to manually create performance-optimized logic and enables nearly immediate visual updates, even across several visualizations. Performance in business settings can be further enhanced with Power BI's support for DirectQuery and Live Connections, which enable scalable connectivity with cloud services and SQL databases.

However, performance can degrade when datasets become exceptionally large or when complex DAX measures and calculated columns accumulate. Users may encounter delays in filtering or visual refreshes under these situations. Furthermore, Power BI desktop functions as a local application unless it is distributed via the Power BI Service, where workspace parameters and user rights can affect performance.

Python Dash, on the other hand, offers more visible control over performance because all of the dashboard's features—from data loading to visual rendering—are handled through code. For medium-sized datasets, Dash can provide good performance when properly optimized with pandas, numpy, and effective callback patterns. However, because data conversions, filtering, and visualization updates are calculated at runtime, the dashboard is more susceptible to changes in dataset size and code quality. Noticeable lag may be introduced by poorly designed callbacks, ineffective data processing, or unfiltered presentation of big datasets.

Nevertheless, Dash gives developers the freedom to apply asynchronous callbacks, server-side pagination, or deferred loading, enabling them to maximize speed for particular use cases. Furthermore, local resource limitations can be lessened by hosting the dashboard on scalable cloud platforms (such as AWS, Heroku, and Dash Enterprise), particularly when several users are engaged at once.

Power BI performed better than Dash in our project in terms of sheer responsiveness, especially when quickly filtering across graphics. Fast



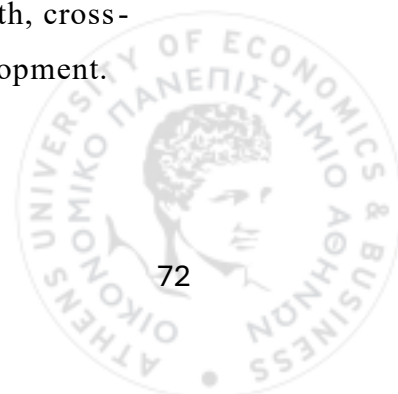
response times can be achieved with minimal tuning of its drag-and-drop model and built-in engine. Even though the Dash version was a little slower in real-time interactions, it still performed well enough for the quantity of the dataset and offered more customizable logic control and data pretreatment, which was useful for analytical activities.

In conclusion, Power BI provides exceptional performance right out of the box, which makes it perfect for quick and easy engagement with complicated or sizable dashboards. Even though it needs to be carefully optimized, Dash offers more flexibility and granular control, which is particularly useful when complex user interactions or custom processing are needed. The best option will rely on whether processing and control flexibility or responsiveness are more important.

5.9 Interactivity and Filtering Logic

Effective dashboard design relies heavily on interaction because it enables users to actively explore data, find insights, and respond to domain-specific queries. Although interactive elements are supported by both Python Dash and Power BI, there are notable differences in their capabilities and implementations.

Power BI's integrated interactivity is excellent. Without knowing any code, users may add slicers, filters, drill-throughs, cross-highlighting, and tooltip details using its user-friendly drag-and-drop interface. Visuals that share data linkages automatically apply interactivity. In a bar chart, for instance, choosing a category instantly filters tables, KPIs, and other charts across the report. Additionally, users may set up visual-, page-, or report-level filters, enable sync slicers across pages, and establish bookmark navigations. Because of this, Power BI is particularly good at providing smooth, cross-component filtering behaviour with no expense in terms of development.



On the other hand, Python Dash necessitates the intentional coding of interactivity through callback functions, which connect user inputs (such as buttons, sliders, and dropdown menus) to output elements (such as tables and graphs). Although it takes more work to design, this enables context-aware and highly configurable filtering. A centralized `filter_data_by_selections` tool, for example, was implemented in this project using Dash. It applies the filters consistently across all tabs and dynamically splits the dataset based on inputs like Scope, Organization, Category, Type, and Year. The `utils/data_filter.py` module implements this logic, and callbacks in the callback file for each tab are used to invoke it.

Additionally, Dash allows for bespoke interactivity, including the ability to control stateful behaviour, update numerous outputs from a single input, and even initiate complex analytics in response to selection changes. However, explicit coding logic and interface planning are needed to get capability comparable to Power BI's native drill-downs or slicers.

Dash's filter persistence across views is one of its main advantages. Selections made in one tab can be readily retained and reflected in another since filtering logic is managed at the data level and shared across tabs via Python modules. Although Power BI provides cross-page filtering with sync slicers, Dash's programmable logic offers greater flexibility than complicated behaviour (such as conditional filtering logic or dynamic updates).

In conclusion, Power BI offers a comprehensive set of integrated interaction tools that are very easy for non-technical users to use and require little setup. It is perfect for quickly creating dynamic dashboards with reliable user interfaces. At the expense of more programming time, Python Dash, on the other hand, provides unmatched flexibility for managing logic and unique, data-driven interactivity. The decision is based on whether complete control and programmability of interactions or speed and simplicity of implementation are more important.

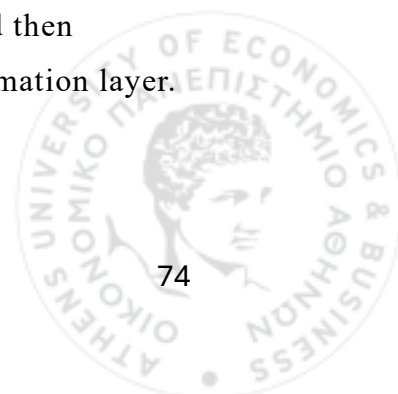


5.10 Integration with External Tools and Pipelines

For visual analytics systems to stay current, reproducible, and scalable, dashboard integration with other tools, automation workflows, and data pipelines is crucial. In this sense, the basic architecture and integration capabilities of Power BI and Python Dash are very different.

Databases, Excel files, SharePoint, APIs, and Azure services are just a few of the many external data sources that Power BI provides out-of-the-box integration with. It allows users to set up load transformations and scheduled refreshes using M code or a graphical user interface through Power Query. Furthermore, Power Automate and Power BI combine to enable workflows like data-driven notifications, email alerts, and scheduled report sharing. However, unless other tools are manually bridged in, Power BI's natural integration capabilities with sophisticated data science workflows, version control systems like Git, and bespoke Python scripts are restricted. Although implementation depends on the platform's environment, Power BI dashboards are usually embedded through the Power BI Service, enabling safe web-based sharing and collaboration. Advanced setups or enterprise-level technologies (such as Power BI REST API, Dataflows, or Premium capacity) are necessary for deep integration with bespoke data pipelines or continuous deployment (CI/CD), even though Microsoft offers some APIs for automation and embedding.

Because the Python Dash dashboard is script-based and flexible by nature, it may be easily customized to fit software engineering workflows and research pipelines. ETL procedures, data validation scripts, machine learning models, and automated reporting may all be readily connected with the application using scheduling technologies like GitHub Actions, Apache Airflow, or Cron tasks. Data can be read directly from any Python-supported source (such as SQL, CSV, or API) by dash apps, preprocessed using pandas, and then dynamically rendered without the need for an additional transformation layer.



Additionally, Dash dashboards may be fully version-controlled using Git, guaranteeing the transparency and repeatability of data manipulations, logic, and visual design. Each component (graphs, callbacks, filters) may be updated or reused independently thanks to the project's modular architecture, and deployment can be completely tailored whether it is hosted on internal servers, Dash Enterprise, Heroku, or Streamlit sharing.

Dash made it possible to integrate reusable Python functions for dynamic filtering, centralized column mapping, and reproducibility metrics in this project with ease. As a result, updating the dashboard in tandem with the underlying dataset was simple. Although Power BI was quicker to connect at first, adding new indicators or columns required more manual changes.

In conclusion, Power BI's ecosystem makes it very efficient for automating simple report refreshes and integrating to popular business data sources. Nonetheless, Python Dash is a more potent option for lengthy, programmatically controlled projects due to its far better integration capabilities for customized data pipelines, research procedures, and automated deployments.



6. Discussion

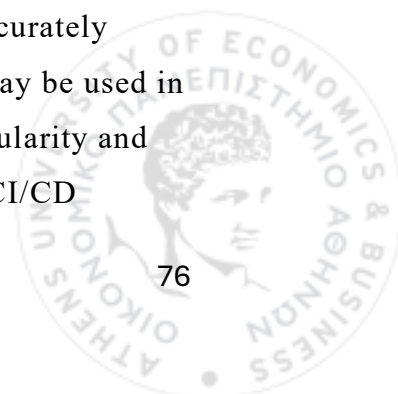
The comparative results of the Power BI and Python dashboard implementations are discussed in this section, along with the advantages and disadvantages of each tool, difficulties experienced during development, and use-case scenarios where each strategy works well.

6.1 Advantages and Limitations of Each Tool

Power BI has many benefits that make it appropriate for creating dashboards quickly and easily. Business users and analysts find its user-friendly drag-and-drop interface especially appealing because it allows for quick prototyping without the need for coding experience. Slicers, treemaps, and KPI cards are examples of built-in visualizations that make data analysis and storytelling more efficient. Because of Power BI's smooth integration with the Microsoft ecosystem, dashboards can be shared via the Power BI Service and embedded into websites, making it extremely accessible to stakeholders that are not technical.

However, because of its low-code paradigm, which prevents fine-grained control over behaviors and visuals, it has limited customization capabilities. Additionally, Power BI dashboards are sensitive to changes in the data structure since they are closely linked to the underlying data model. Furthermore, repeatability in dynamic or research-intensive situations is limited by Power BI's absence of native automation features and version control.

Python (Plotly Dash + VS Code), on the other hand, offers complete code customisation. Layout, style, logic, and interaction may all be accurately controlled by developers. With reusable parts and features that may be used in different applications, the dashboard framework encourages modularity and reusability. Additionally, Dash easily interfaces with databases, CI/CD



pipelines, APIs, and machine learning workflows.

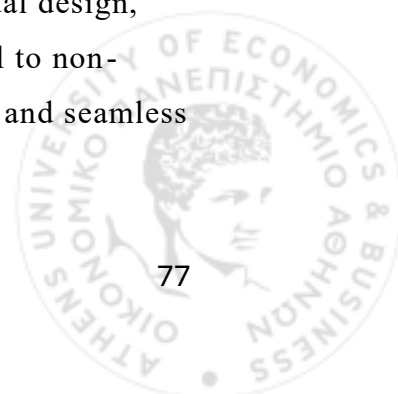
The primary disadvantages are the requirement for knowledge of web development techniques and proficiency with Python programming. In the absence of a well-thought-out implementation, stakeholders can find the UI less user-friendly. In comparison to Power BI's out-of-the-box environment, the setup procedure, which involves Visual Studio Code and package management, can also be more complicated.

6.2 Tool Suitability - Ideal Use Cases

Table 2: Tool Recommendation Based on Project Scenario

Scenario	Recommended Tool
Rapid prototyping for business users	Power BI
Sharing reports with non-technical audiences	Power BI
Projects with a stable schema and high design consistency	Power BI
Projects requiring automation, versioning, or reproducibility	Python (Dash)
Scientific workflows or integration with ML models	Python (Dash)
Projects expected to evolve with changing data structure	Python (Dash)

The final decision to deploy Power BI as the primary solution for the research center's public-facing dashboards was driven by its polished visual design, ease of web-based sharing, and intuitive interface that caters well to non-technical users. Its out-of-the-box visuals, consistent formatting, and seamless



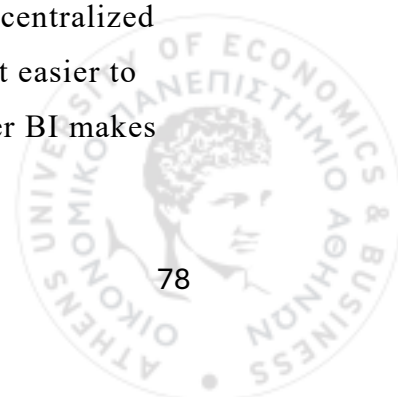
integration within the Microsoft ecosystem made it particularly suitable for stakeholder communication and institutional visibility.

However, the parallel Python implementation using Plotly Dash highlighted clear strengths in terms of adaptability, modular development, and reproducibility. Its code-based architecture, use of centralized configuration (e.g., `COLUMN_MAP`), and compatibility with automation tools make it especially valuable for internal use, future feature expansion, and integration with scientific or data science workflows. Although not selected for deployment in this instance, the Python dashboard remains a robust and scalable alternative—offering a reusable foundation for future projects, research extensions, or programmatic use cases within the institution.

6.3 Insights and Challenges During Implementation

Maintaining feature parity between the two tools was challenging. It took careful preparation and specialized scripts to replicate Power BI's visual elements, such as slicers and KPI cards, in Dash. However, a more customized and reusable interface was made possible by this development effort. The dashboard's visual style closely matched that of the Power BI version, with a white background, rounded KPI cards with shadows, and high contrast color schemes (red for citation metrics, blue for reproducibility, etc.). Modular Python callbacks (such as `overview_callbacks.py` and `geo_callbacks.py`) provided fine-grained control over filtering, tooltips, and dynamic outputs, enabling Dash's interactivity.

Maintainability was greatly improved by segmenting the Python dashboard into reusable modules (such as `kpi_section.py`, `selection_boxes.py`, and shared graph methods). These elements guaranteed design coherence across tabs and enabled quick upgrades. One significant benefit was the use of a centralized `COLUMN_MAP` to handle data schema variability, which made it easier to adjust to renamed or restructured columns—something that Power BI makes more challenging.



Each tool had a different performance. Mid-sized datasets were handled effectively by Power BI, particularly for default interactions. Python scaled well for complicated filters and transformations, especially when combined with pandas and efficient callbacks, while being slower to configure at first. These encounters reaffirmed how crucial it is to match tool selection to long-term technical objectives as well as stakeholder expectations.



7. Conclusion & Recommendations

7.1 Summary of Findings

In this thesis, the Plotly Dash framework was used to compare the implementation of a research evaluation dashboard in Python and Power BI. Evaluating each tool's performance along several important characteristics, including visual quality, interactivity, flexibility, scalability, and appropriateness for changing datasets, was the aim.

The Power BI dashboard was excellent at offering a dynamic, aesthetically pleasing interface that non-technical stakeholders could share and understand with ease. Its integrated KPI cards, slicers, and designing tools facilitated efficient communication and quick development. However, Power BI showed shortcomings in terms of adaptation to changes in data structures, automation, and modularity.

However, the Python-based dashboard provided more control over data processing, layout customization, and application logic. Long-term maintainability and flexibility were facilitated by its modular design, use of reusable components, and centralized configuration using tools like `COLUMN_MAP`. These advantages did, however, come at the expense of more complicated development. A strong grasp of Python, web technologies, and software engineering best practices were necessary for setting up the environment, creating and testing callbacks, and overseeing the dashboard layout. This method works well for internal use cases or research where reproducibility, version control, and flexibility in response to evolving datasets are essential, but it might not be feasible for teams lacking technical know-how.



7.2 Recommendation: When to Use Power BI vs. Python Dash

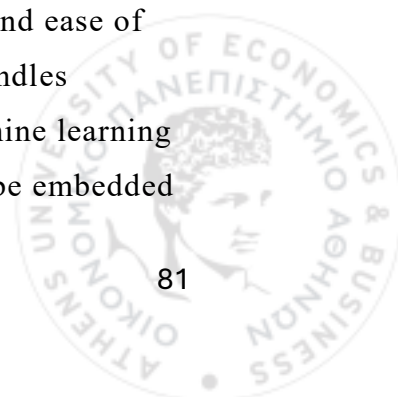
Because of its user-friendliness and professional visual output, Power BI was finally selected for implementation on the research center's websites given the project's context. Nonetheless, the Python dashboard offers a strong basis for future development, such as interaction with machine learning-based assessment systems, automated reporting, or programmatic pipelines.

Table 3: Recommended Tool Selection Based on Long-Term Objectives

Scenario	Preferred Tool
Short-term business reporting with fixed schema	Power BI
Rapid dashboarding for non-technical teams	Power BI
Public sharing and embedding on websites	Power BI
Custom scientific dashboards with evolving metrics	Python (Dash)
Integration with ML workflows or APIs	Python (Dash)
Reproducible and version-controlled environments	Python (Dash)

7.3 Suggestions for Future Work

To further strengthen the comparative research and enhance the practical utility of the dashboard implementations, several future directions can be explored. First, a hybrid deployment strategy could be adopted to combine the strengths of both platforms. Power BI, with its polished visuals and ease of presentation, can be used as the front-end layer, while Python handles complex backend processes such as real-time data analysis, machine learning predictions, or reproducibility scoring. These results could then be embedded



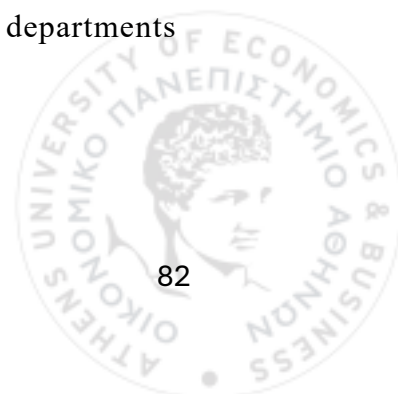
into Power BI reports through APIs or data connectors, offering a seamless integration between analytical depth and communicative clarity [34].

Second, the user interface of the Python dashboard could be significantly improved using modern component libraries such as Dash Mantine Components, Dash Bootstrap, or other community-developed extensions. These tools offer enhanced styling, responsiveness, and layout control, which can help close the visual and usability gap between Python-based dashboards and Power BI. Exploring such improvements would not only increase user adoption but also make Python dashboards more suitable for broader audiences beyond technical users.

Third, addressing the lack of version control in Power BI is a key area for improvement. While Python development naturally integrates with tools like Git, Power BI lacks built-in support for collaborative versioning. Future work could investigate best practices or third-party tools that enable structured change tracking, collaborative editing, and documentation in Power BI environments, especially for teams working across multiple reports or datasets [35].

Fourth, more thorough testing of scalability and performance is needed. Future research could include stress-testing the dashboards with higher data volumes, more intricate user interactions, and numerous concurrent users, even though both tools handled the datasets used in this study well. Insights about memory usage, rendering speed, and response time under load—all crucial elements for institutional or enterprise deployment—would be obtained from this.

Fifth, a strong deployment plan for the Python dashboard might be put into action. The application can be deployed on cloud platforms like Heroku, Azure Web Apps, or AWS Elastic Beanstalk, containerized using Docker, or embedded in JupyterHub settings for research teams. This kind of implementation would facilitate cooperative use amongst several departments or institutions, improve accessibility, and expedite updates.



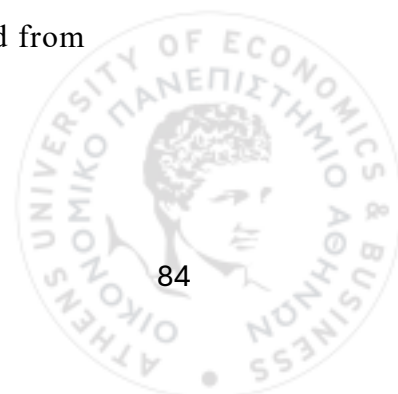
Lastly, the dashboard's impact could be further increased by extending its functionality to include multilingual capabilities, cross-institutional comparisons, or connection with data governance systems. In sensitive research settings, features like user roles, audit trails, and privacy-preserving analytics may be crucial, particularly when dashboards are shared across public platforms or geographical locations.

To sum up, these future prospects provide distinct avenues for improving dashboarding technologies' scalability, usability, and adaptability. They also highlight how important it is to match dashboard design to the changing requirements of stakeholder involvement, institutional openness, and data-driven research.

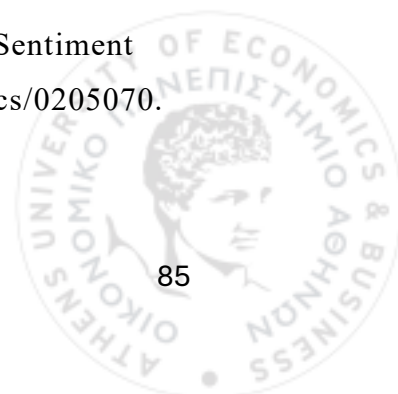


8. References

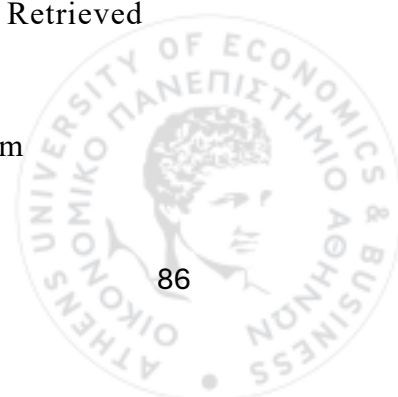
- [1] Smith, V. S. (2013). Data dashboard as evaluation and research communication tool. *New directions for evaluation*, 2013(140), 21-45.
- [2] Knafllic, C. N. (2015). *Storytelling with data: A data visualization guide for business professionals*. John Wiley & Sons.
- [3] Powell, B. (2017). *Microsoft Power BI cookbook: Creating business intelligence solutions of analytical data models, reports, and dashboards*. Packt Publishing Ltd.
- [4] Baars, H., & Kemper, H. G. (2008). Management support with structured and unstructured data—an integrated business intelligence framework. *Information systems management*, 25(2), 132-148.
- [5] Sievert, C. (2020). *Interactive web-based data visualization with R, plotly, and shiny*. Chapman and Hall/CRC.
- [6] Ergasheva, S., Ivanov, V., Khomyakov, I., Kruglov, A., Strugar, D., & Succi, G. (2020, May). InnoMetrics dashboard: The design, and implementation of the adaptable dashboard for energy-efficient applications using open source tools. In *IFIP International Conference on Open Source Systems* (pp. 163-176). Cham: Springer International Publishing.
- [7] Kolokolov, A., & Zelensky, M. (2024). *Data visualization with Microsoft power BI*. " O'Reilly Media, Inc."
- [8] Microsoft. (2024). *Power Query Documentation*. Retrieved from <https://docs.microsoft.com/power-query/>
- [9] Microsoft. (2024). *DAX Reference*. Retrieved from <https://docs.microsoft.com/dax/>
- [10] Plotly Technologies Inc. (2023). *Dash User Guide*. Retrieved from <https://dash.plotly.com/>



- [11] Plotly Technologies Inc. (2023). Plotly Express. Retrieved from <https://plotly.com/python/plotly-express/>
- [12] Dash Bootstrap Components. (2022). Dash Bootstrap Components Documentation. Retrieved from <https://dash-bootstrap-components.opensource.faculty.ai/>
- [13] McKinney, W. (2010). Data structures for statistical computing in Python. *scipy*, 445(1), 51-56.
- [14] Microsoft. (2023). Visual Studio Code. Retrieved from <https://code.visualstudio.com/>
- [15] Chacon, S., & Straub, B. (2014). *Pro Git*, 2nd edn. Apress, Berkeley.
- [16] Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., ... & Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data*, 3(1), 1-9.
- [17] Kotitsas, S., Pappas, D., Manola, N., & Papageorgiou, H. (2023). SCINOBO: a novel system classifying scholarly communication in a dynamically constructed hierarchical Field-of-Science taxonomy. *Frontiers in Research Metrics and Analytics*, 8, 1149834.
- [18] Scelles, N., & Teixeira da Silva, J. A. (2025). Making the impact of publications within a field comparable by improving the field-weighted citation impact (FWCI): the case of sport management. *Scientometrics*, 130(3), 1571-1586.
- [19] Teufel, S., Siddharthan, A., & Tidhar, D. (2006, July). Automatic classification of citation function. In *Proceedings of the 2006 conference on empirical methods in natural language processing* (pp. 103-110).
- [20] Cronin, B. (1984). The citation process. The role and significance of citations in scientific communication, 103.
- [21] Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up? Sentiment classification using machine learning techniques. arXiv preprint [cs/0205070](https://arxiv.org/abs/cs/0205070).



- [22] Microsoft. (2023). Microsoft Excel. Retrieved from <https://www.microsoft.com/excel>
- [23] Kandel, S., Paepcke, A., Hellerstein, J. M., & Heer, J. (2011). Wrangler: interactive visual specification of data transformation scripts. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 3363–3372.
- [24] Nielsen, J., & Molich, R. (1990). Heuristic evaluation of user interfaces. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 249–256.
- [25] Few, S. (2006). Information Dashboard Design: The Effective Visual Communication of Data. O'Reilly Media.
- [26] Souders, S. (2007). High Performance Web Sites: Essential Knowledge for Front-End Engineers. O'Reilly Media.
- [27] Peng, R. D. (2011). Reproducible research in computational science. Science, 334(6060), 1226–1227.
- [28] Fellows, S. A. (2021). Python Dash for Data Visualization: A Guide to Building Interactive Dashboards. Addison-Wesley.
- [29] Walkenbach, J. (2018). Excel Data Analysis: Your Visual Blueprint for Analyzing Data, Charts, and PivotTables. Wiley.
- [30] Grinberg, M. (2018). Flask web development. " O'Reilly Media, Inc.".
- [31] Bach, B., Freeman, E., Abdul-Rahman, A., Turkay, C., Khan, S., Fan, Y., & Chen, M. (2022). Dashboard design patterns. IEEE transactions on visualization and computer graphics, 29(1), 342-352.
- [32] VanderPlas, J. (2016). Python data science handbook: Essential tools for working with data. " O'Reilly Media, Inc.".
- [33] Brewer, C. A. (2003). ColorBrewer: Color Advice for Maps. Retrieved from <https://colorbrewer2.org/>
- [34] Microsoft. (2024). Use Python with Power BI. Retrieved from <https://docs.microsoft.com/power-bi/connect-data/python-scripts>



[35] Microsoft. (2024). Power BI DevOps and Application Lifecycle Management. Retrieved from <https://docs.microsoft.com/power-bi/guidance/devops-and-lifecycle-management>

