PyGWalker: On-the-fly Assistant for Exploratory Visual Data Analysis

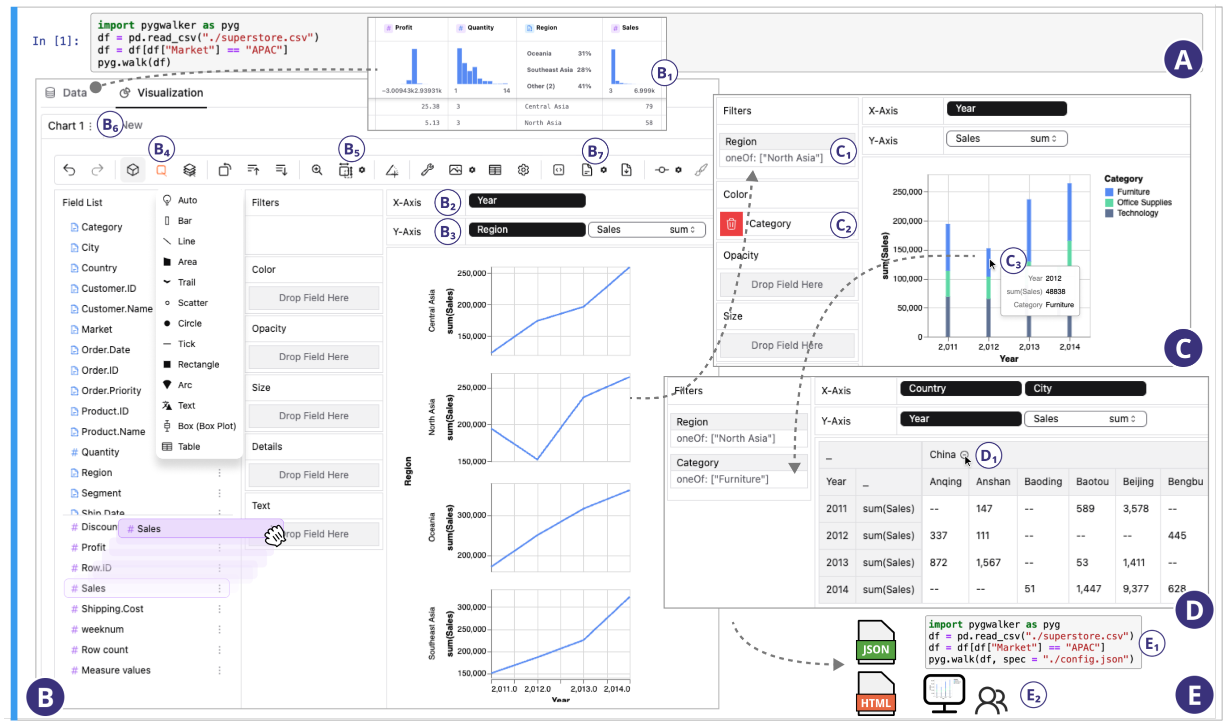
Yue Yu[*]
Hong Kong University of Science and Technology
Kanaries Data

Leixian Shen[†]
Hong Kong University of Science and Technology

Fei Long[‡]
Kanaries Data

Huamin Qu[§]
Hong Kong University of Science and Technology

Hao Chen[¶]
Kanaries Data



```
In [1]: import pygwalker as pyg
        df = pd.read_csv("./superstore.csv")
        df = df[df["Market"] == "APAC"]
        pyg.walk(df)
```

_		China					
Year	_	Anqing	Anshan	Baoding	Baotou	Beijing	Bengbu
2011	sum(Sales)	--	147	--	589	3,578	--
2012	sum(Sales)	337	111	--	--	--	445
2013	sum(Sales)	872	1,567	--	53	1,411	--
2014	sum(Sales)	--	--	51	1,447	9,377	628

```
import pygwalker as pyg
df = pd.read_csv("./superstore.csv")
df = df[df["Market"] == "APAC"]
pyg.walk(df, spec = "./config.json")
```

Figure 1: An example walkthrough with PyGWalker in a Jupyter Notebook. The analyst initially loads the dataset into PyGWalker to activate the interface with one line of code (A). The dataset overview is provided under the Data tab (B_1). Under the Visualization tab, the analyst creates multi-faceted charts (B) by dragging data columns to the X-Axis (B_2) and Y-Axis (B_3) and selecting the mark type (B_4). Filters are further added to narrow the dataset (C_1), and the color encoding (C_2) can be diversified for varied visualizations (C). Details can be viewed by hovering over chart elements (C_3). In addition to charts, pivot tables can be constructed (D) with collapsible hierarchical headers (D_1). Finally, the visualizations generated in the exploratory analysis process can be documented (E). The analyst adjusted and chart style (B_5), renamed charts (B_6), and exported them (B_7) as a JSON specification (E_1) and an HTML file of the notebook (E_2), which can be shared and hosted to reproduce the analysis process (E_2), accessible at this link.

Abstract

Exploratory visual data analysis tools empower data analysts to efficiently and intuitively explore data insights throughout the entire analysis cycle. However, the gap between common programmatic analysis (e.g., within computational notebooks) and exploratory visual analysis leads to a disjointed and inefficient data analysis experience. To bridge this gap, we developed PyGWalker, a Python library that offers on-the-fly assistance for exploratory visual data analysis. It features a lightweight and intuitive GUI with a shelf builder modality. Its loosely coupled architecture supports multiple computational environments to accommodate varying data sizes. Since its release in February 2023, PyGWalker has gained much attention, with 612k downloads on PyPI and over 10.5k stars on GitHub as of June 2024. This demonstrates its value to the data science and visualization community, with researchers and developers integrating it into their own applications and studies.

Index Terms: Human-centered computing — Visualization — Visualization systems and tools

1 Introduction

Exploratory visual data analysis tools can help data analysts effectively and intuitively explore data insights and guide further analyses. This process is iterative and spans the entire data analysis cycle. However, when analysts engage in programming, such as within computational notebooks, numerous data variables are generated. Mapping these variables into visual representations that align with the user's intentions and supporting exploratory visual analysis is non-trivial, especially for those who are unfamiliar with visual design. Analysts are often faced with the choice of either exporting the data and importing it into existing exploratory visual data anal-

[*]e-mail: yue.yu@connect.ust.hk. This work was done during Yue Yu's internship at Kanaries Data.
[†]e-mail: lshenaj@connect.ust.hk
[‡]e-mail: feilong@kanaries.net
[§]e-mail: huamin@cse.ust.hk
[¶]e-mail: haochen@kanaries.net. Hao Chen is the corresponding author.

ysis tools (e.g., Voyager [40] and DeeyEye [16]) or relying on complex programming to repeatedly visualize their data. The gap between programmatic analysis and exploratory visual analysis leads to a disjointed data analysis experience.

To address these issues, we developed PyGWalker, a Python library that offers on-the-fly assistant for exploratory visual data analysis in computational notebooks, available as open source at github.com/Kanaries/pygwalker. PyGWalker enables users to effortlessly invoke a lightweight exploratory visual data analysis tool with just a single line of code, as shown in Fig. 1. The intuitive graphic user interface (GUI) features a shelf builder modality [28], allowing users to intuitively drag and drop variables onto visual channels and dynamically experiment with various visual representations. Furthermore, its architecture, which decouples interaction, computation, and rendering, supports the integration of multiple computational environments, such as JavaScript, Python kernel, and external databases, to accommodate varying data sizes.

PyGWalker's rapid adoption underscores its value to the data science and visualization community. Community members have enthusiastically contributed tutorials and demos [3, 5, 6, 13, 19, 24], with researchers and developers integrating PyGWalker into their own applications and studies.

2 Related Work

PyGWalker draws upon existing works about exploratory visual data analysis and assistance in computational notebooks.

Exploratory Visual Data Analysis: Data analysis often involves users continuously exploring insights through iterations and drilling down. Existing exploratory visual data analysis tools can help users effectively and intuitively complete these tasks. For example, Voyager [40] uses statistical and perceptual measures to recommend suitable charts and supports faceted browsing. MEDLEY [25] and TaskVis [31, 32] support user-intention-driven visual data analysis. DMiner [11] and MultiVision [41] automatically generate multi-view dashboards based on user data. However, these are all independent analysis systems. The separation between coding and exploratory visual data analysis requires users to switch between two modes, hindering a seamless data analysis process.

Assistance in Computational Notebooks: Computational notebooks are a popular data analysis environment, and many studies have developed corresponding assistive features for visual analysis and storytelling. For example, Lux [9] is an always-on framework that helps users quickly preview insights in their data. Notable [10] can automatically convert data facts that users are interested in during their analysis process into slides to facilitate data communication. InkSight [12] allows users to sketch interesting data insights directly on visualizations and then uses LLMs to generate descriptions. B2 [42] uses data queries as a shared representation to bridge the gap between code and interactive visualizations, automatically generating dashboards. Although these tools can quickly help users generate data analysis and communication results, they all lack support for on-the-fly process-oriented exploratory visual analysis.

3 PyGWalker

In this section, we will first give an overview of PyGWalker, and then walk through a usage scenario to demonstrate the interaction workflow for users. Next, we will discuss the computation engine behind PyGWalker and introduce the rendering of visualizations.

3.1 Overview

The main workflow is as follows: users can easily insert a line of code to call PyGWalker during data analysis, which will activate the interactive GUI (named Graphic Walker) and automatically load the relevant data. To represent the user's intent information and data information during interactions on the GUI, we have designed a declarative scheme, called Graphic-Link (Sec. 3.3).

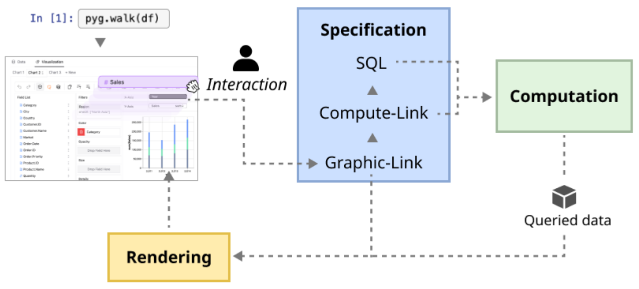


Figure 2: Overview of PyGWalker.

Furthermore, to support various computing paradigms and environments (e.g., browsers, databases, cloud computing, etc.), we have also derived a computation-driven scheme, called Compute-Link (Sec. 3.4), from Graphic-Link, aiming to decouple view data processing from visualization specifications. Compute-Link allows the generating of user-intent-based view data or SQL queries. The aggregated data from these queries, as well as Graphic-Link, is then rendered into interactive visualizations in real time. As the user iteratively conducts exploratory visual analysis on Graphic Walker, the corresponding specification is continuously updated.

3.2 Usage Scenario

Imagine Alice, a data analyst in a superstore enterprise, is assigned a task to analyze the global superstore dataset [18]. Fig. 1 illustrates how she integrates PyGWalker into her programming workflow to seamlessly conduct exploratory visual data analysis.

In her Jupyter Notebook, Alice begins by importing the global sales dataset from a CSV file, applying a filter to narrow her analysis to the Asia Pacific market. She then initiates PyGWalker by executing `pyg.walk(df)` within her notebook (A), and PyGWalker's interactive GUI directly appears in the output cell. After glancing over the dataset overview under the Data tab (B_1), she turns to the Visualization tab and starts her exploration. Alice is interested in the sales over years of different regions for a broad picture. Therefore, she drags the *Year* field from the Field List and drops to the X-Axis shelf (B_2), and *Region* and Sum of *Sales* fields to the Y-Axis shelf (B_3), using the *Line* mark type (B_4). The interface responds by displaying four line charts (B), faceted by region, showing a significant drop in sales for North Asia in 2012, prompting Alice to investigate this anomaly more deeply.

Alice thus removes the *Region* from the X-Axis and adds a filter by dragging the *Region* field into the Filters panel, filtering for "North Asia" (C_1). To explore sales by category, she drags the *Category* field to the Color encoding shelf (C_2) and switches the mark type to *Bar*. The resulting bar chart (C) shows a notably sharp decline in the blue segment in 2012. Hovering on the segment (C_3), the tooltip shows the segment is the "Furniture" category.

Curious about the specific cities contributing to the decline, Alice shifts to a more granular-level analysis by examining the exact numbers of furniture sales across different cities over the years. She applies a filter for "Furniture", sets the mark type to Table, and organizes the data hierarchically with *Year* on the Y-Axis and *Country* and *City* on the X-Axis. This creates a pivot table that groups cities within their countries by year (D). As there are hundreds of cities, she first collapses the country-level headers (D_1) and notes that between 2011 and 2012, sales in China and South Korea dropped significantly. By expanding the headers of these countries, she identifies major cities like Beijing, Jining, and Seoul, which showed drastic sales declines from robust figures in 2011 to near-zero in 2012, pinpointing them as the primary sources of the downturn.

Alice finally documents and shares her findings (E). After adjusting the chart style (B_5) and renaming the chart tabs (B_6), she exports the visualization specifications into a JSON file `config.json` (B_7) and shares it with her colleagues responsible for the supply chain in the affected cities. They can replicate her visualizations by loading the specification using `pyg.walk(df, spec="./config.json")` in their own notebooks (E_1). Additionally, Alice exports the entire notebook as an HTML file[1] and hosts it online, enabling colleagues without a programming environment to view her exploration steps in different tabs and interact with her visualizations (E_2).

3.3 Information Representation

After the user loads their data with PyGWalker and interacts on the Graphic Walker GUI, the data and user's analysis intents through interactions are stored in a declarative specification, named Graphic-Link[2]. It mainly encodes the following primitive types:

Data Characteristic: Graphic-Link encodes data columns and their automatically inferred semantic information. Each data column is assigned a semantic data type (nominal, ordinal, or quantitative) and an analytic type, which is either a "dimension" (categorical descriptors that segment data) or a "measure" (quantifiable metrics) [8]. This helps that visualizations faithfully depict the inherent relationships present in the data.

Data Transformation: Graphic-Link keeps a record of users' data transformation operations. Users can interactively define filtering criteria, enabling the specification of value sets or ranges for data columns. Moreover, users can explicitly specify aggregation functions such as "sum" and "average", and performing field transformations (such as logarithms or binning) will result in a new transformed data column. Users can also sort and stack their data with different styles.

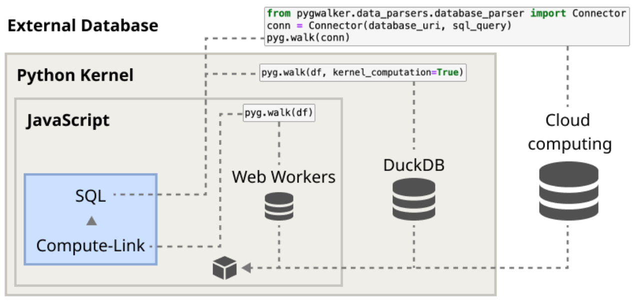
Visual Encoding: Graphic-Link captures and records users' interactions with the GUI, which represents their intentions for visual data analysis. Users can define the fundamental geometric primitives employed in the visualization, such as "line," "bar," "point," and more. Additionally, to enhance the exploration process, a default mark type is automatically generated based on heuristic rules inspired by existing automatic visualization creation systems [17, 40]. Furthermore, users can intuitively assign their data columns to various visual channels, including the X-axis, Y-axis, color, size, shape, opacity, and more, by simply dragging and dropping them. Furthermore, it supports automatic faceting when multiple dimensions are hierarchically organized along an axis.

Style and Configuration: To facilitate cross-platform viewing and sharing, Graphic-Link also encodes additional configurations related to coordinate systems (chart or geographic), layout modes, chart styles, scale ranges, color palettes, etc. This ensures that visualizations remain consistent and accurate across different platforms and devices, preserving the intended visual aesthetics.

Graphic-Link provides a comprehensive representation of data characteristics and user interaction information, establishing a foundation for subsequent computation and rendering processes.

3.4 View Data Computation

To enable adaptability across diverse computational paradigms, we additionally introduce Compute-Link[3], a computation-oriented specification that separates view data processing from visualization specifications. While Graphic-Link aims to declare the visualization contents, Compute-Link extracts the necessary view data for a given visualization from the comprehensive Graphic-Link specification. The derivation from Graphic-Link to Compute-Link mirrors the user's process of mapping data columns to visual channels in the GUI. This transformation is guided by algebraic rules in the



```
from pygwalker.data_parsers.database_parser import Connector
conn = Connector(database_uri, sql_query)
pyg.walk(conn)
```

```
pyg.walk(df, kernel_computation=True)
```

```
pyg.walk(df)
```

Figure 3: Computation mechanisms of PyGWalker.

Grammar of Graphics [39] (i.e., cross, nest, and blend operators), which determines how different data attributes are combined and represented to compute the view data, particularly for constructing pivot tables and multi-faceted charts.

Compute-Link consists of a series of data queries that describe how to compute the view data from the raw data: *Filter Query* selects specific data subsets based on users' defined criteria; *Transform Query* applies calculations and transformations to the data columns, such as calculating the logarithm or binning; *View Query* structures the data into the desired format for visualization, which can be aggregated by different operations or directly structured as raw; and *Sort Query* orders the result data.

This separation not only maintains a clear decouple from the visualization specification but flexibly supports a wide array of computational modes and environments such as web browsers, databases, and cloud computing platforms, as illustrated in Fig. 3. The following illustrates some computational scenarios:

JavaScript Computation: By default, when a user executes `pyg.walk(df)`, the computation occurs within the JavaScript environment of the application, utilizing a pipeline of web workers for asynchronous data processing. These workers parse and execute the series of operations defined in Compute-Link, ultimately producing the finalized view data.

Python Kernel Computation: While JavaScript computation may be limited to small-scale computation, users can opt for Python kernel computation by setting `kernel_computation=True` for more substantial datasets that exceed the capabilities of JavaScript. This mode parses Compute-Link specifications into analytical SQL queries to be processed by the DuckDB engine [27], an open-source analytic database with high computation performance, in the Python Kernel. The results are then conveyed back to the JavaScript environment for rendering.

External Database Computation: For users who need to analyze large datasets stored in non-local databases or online data warehouses, PyGWalker also offers an external computation capability. To utilize this functionality, users can import `Connector` class from the `database_parser` module and create a Connector object (e.g., `conn`) by supplying the SQLAlchemy connection URI and a SQL query to retrieve the dataset. Once configured, the analysis can be conducted directly on the external database by executing `pyg.walk(conn)`. Similar to kernel computation, analytical SQL queries will be generated and sent to the external database, and the results are then fetched and passed to the JavaScript environment.

3.5 Visualization Rendering

Graphic-Link specifications, combined with the computed view data, are finally translated into visual representations, which support the rendering of different visualization types: *General Charts:* For common chart types (e.g., lines, bars, points, etc.), Graphic-Link is initially converted into a Vega-Lite specification [29], leveraging the Vega-Lite rendering engine for its expressiveness, widespread usage, and adherence to the principles of the Grammar of Graphics [39]. *Pivot Tables:* When a table mark type is

[1]https://kanaries.net/gallery/pygwalker/notebooks/superstore.html

[2]https://graphic-walker.kanaries.net/chartinfo.json

[3]https://github.com/Kanaries/graphic-walker/blob/main/computation.md

chosen, a pivot table is directly rendered to effectively display aggregated data. *Geographic Visualizations:* In the case of drawing a geographic visualization, an OpenStreetMap base layer will be established, overlaid by encodings that are mapped to longitude and latitude, ensuring accurate spatial representation of data elements.

3.6 Cross-Platform

In addition to providing support for Jupyter Notebooks, PyGWalker expands its capabilities to cater to data analysts working on different platforms and utilizing various programming languages.

PyGWalker seamlessly integrates with popular notebook-based analysis environments like Colab and Kaggle Notebook, promoting collaboration and knowledge sharing. Furthermore, PyGWalker can also be embedded into Python frameworks that support HTML rendering, such as Streamlit [35] and Gradio [1]. This empowers developers to create bespoke visualization analysis interfaces within their own applications swiftly.

Owing to the flexible design of the architecture of PyGWalker, the interactive GUI, Graphic Walker[4], can be viewed as a standalone TypeScript-based React library to enable flexible integration into other environments, such as web applications and other programming languages. For example, recognizing the diverse data science landscape, we've developed a similar package for the R programming environment[5]. This opens the door to exploratory visual analysis for a large community of R-focused data scientists.

4 Release and Usage

PyGWalker [6] was first released and open-sourced in February 2023. As of June 2024, it has gained significant popularity, with over 612k total downloads, approximately 90k monthly downloads, and over 10.5k stars on GitHub. These quantitative metrics indicate that PyGWalker has established a strong presence in the developer community. To delve deeper into PyGWalker's real-world impact, we systematically analyzed how developers and researchers utilize the tool. Our findings are based on two key sources: (1) Metadata from 536 public GitHub repositories listing PyGWalker as a dependency and (2) 7 research articles on Google Scholar containing the keyword "PyGWalker" in their works.

The analysis of GitHub repositories revealed some interesting insights. For example, among 536 repositories using PyGWalker, 358 (66.8%) are Python applications, 138 (25.7%) are Jupyter Notebooks, and the remaining 40 (7.5%) utilize other languages like HTML and JavaScript. We also found 101 repositories (18.8%) explicitly mentioning frameworks like "Streamlit," "Django," and "Flask". This suggests PyGWalker's adaptability for seamless integration into web applications, extending its value to the open-source developer community beyond traditional data analysis workflows. Looking into some detailed cases, we found developers typically leverage PyGWalker as an out-of-the-box exploratory dashboard to support dynamic data visualization. For example, BICat [37] is a Streamlit-based application that integrates PyGWalker and ChatGPT [23] to enable users to interactively explore an uploaded dataset by natural language. Similarly, Diagnostic Expert Advisor [26] juxtaposes PyGWalker with predefined plots, offering users both guided views and the freedom of self-directed exploration. These findings underscore how PyGWalker's convenient and seamless integration empowers developers to build web applications moving beyond static visualizations.

While we understand GitHub may not fully reflect the extent of PyGWalker's use in notebooks due to data privacy considerations, we found compelling evidence of its adoption in research. Researchers from diverse fields, including meteorology [7], traffic [14], biomechanics [20], and waste management [22], have embraced PyGWalker, showcasing its potential effectiveness across disciplines. Besides research applications, PyGWalker is also introduced in data science education programs. For instance, the book chapter *Data Visualization for Business Intelligence* [36] features PyGWalker for data visualization in business intelligence, stating it *"simplifies the data analysis and data visualization workflow."* Furthermore, Python crash courses like better-py/learn-py [2] integrate PyGWalker, emphasizing its user-friendly nature and low-code approach. These cases show PyGWalker's capabilities to enable learners to explore datasets quickly and gain visual insights, enhancing their data analysis skills.

5 Discussion

We discuss the limitations of PyGWalker and potential future directions to enhance it.

Expand Visualization Capabilities: PyGWalker's architecture is intentionally designed to separate visualization rendering from information representation and data processing, allowing flexible integration of diverse rendering libraries and engines. Currently, PyGWalker primarily relies on Vega-Lite, benefiting from its expressiveness and ease of use. However, we acknowledge that for specialized or interactive visualizations with complex requirements, a lower-level library like D3 [4] may be indispensable. In the future, we plan to empower users with greater customization and control, allowing them to create a wider range of visual narratives with their data, including support for more visualization types [33] and narrative types [34].

Design Intelligent Interactions: While our current interaction design is intuitive, it does require users to have some understanding of the tool-specific operations to produce effective visualizations. One promising direction is integrating natural language interfaces with the help of large language models (LLMs), allowing users to freely talk their analysis intents to the system and directly generate corresponding results [30]. Furthermore, PyGWalker's interactive interface provides opportunities for users to explore and refine the visualizations generated by LLMs [38], creating a feedback loop. To leverage these capabilities, we are actively developing a feature that translates natural language descriptions and dataset metadata into Graphic-Link specifications.

Leverage PyGWalker as a Learning Tool: Previous research underscores the benefits of interactive notebooks for teaching data visualization over traditional slides [15], shedding light on the potential for PyGWalker as an effective education tool. Currently, PyGWalker only focuses on exporting the final visualization output for further use. In the future, we plan to enhance the analytic provenance that captures the visualization creation process [21]. This would allow educators to share step-by-step walkthroughs of their visual data analysis workflows, enabling students to learn by actively retracing the expert's reasoning and experimentation.

6 Conclusion

This paper proposes PyGWalker, which bridges the gap between programmatic analysis and exploratory visual analysis tools. Its user-friendly GUI and loosely coupled architecture have gained considerable attention from the data science and visualization community, with users actively contributing tutorials and developers integrating it into their own applications. We will continually enhance and expand the tool's functionality, contributing to the open-source community. We also encourage more users to join in its development and inspire further interesting and practical research.

Acknowledgments

The authors wish to thank all of the contributors from our open-source community who support PyGWalker and related projects.

[4]https://github.com/Kanaries/graphic-walker
[5]https://github.com/Kanaries/GWalkR
[6]https://github.com/Kanaries/pygwalker

References

[1] A. Abid, A. Abdalla, A. Abid, D. Khan, A. Alfozan, and J. Zou. Gradio: Hassle-free sharing and testing of ML models in the wild, June 2019. 4

[2] better py. learn-py. https://github.com/better-py/learn-py, 2024. [last accessed 16 June 2024]. 4

[3] S. Bosau. A tableau alternative in python for data analysis (in streamlit & jupyter) — pygwalker tutorial. https://www.youtube.com/watch?v=Ynt7Etci1KU. [last accessed 16 June 2024]. 2

[4] M. Bostock, V. Ogievetsky, and J. Heer. D³ data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011. 4

[5] BugBytes. Pygwalker - python data visualization tool / streamlit integration. https://www.youtube.com/watch?v=ogyxjkYRgPE. [last accessed 16 June 2024]. 2

[6] M. Das. Pygwalker: A graphic walkthrough of tableau-style user interface in jupyter (python). https://medium.com/@HeCanThink/pygwalker-a-graphic-walkthrough-of-tableau-style-user-interface-in-jupyter-python-15674be950bb. [last accessed 16 June 2024]. 2

[7] M. ElTaweel, S. Alfaro, G. Siour, A. Coman, S. Robaa, and M. A. Wahab. Prediction and forecast of surface wind using ml tree-based algorithms. *Meteorology and Atmospheric Physics*, 136(1):1, 2024. 4

[8] P. Janus and G. Fouché. *Cubes, Dimensions, and Measures*, pp. 15–39. Apress, Berkeley, CA, 2009. 3

[9] D. J.-L. Lee, D. Tang, K. Agarwal, T. Boonmark, C. Chen, J. Kang, U. Mukhopadhyay, J. Song, M. Yong, M. A. Hearst, and A. G. Parameswaran. Lux: Always-on Visualization Recommendations for Exploratory Dataframe Workflows. *Proceedings of the VLDB Endowment*, 15(3):727–738, 2021. 2

[10] H. Li, L. Ying, H. Zhang, Y. Wu, H. Qu, and Y. Wang. Notable: On-the-fly Assistant for Data Storytelling in Computational Notebooks. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, CHI'23*, pp. 1–16. ACM, 2023. 2

[11] Y. Lin, H. Li, A. Wu, Y. Wang, and H. Qu. DMiner: Dashboard Design Mining and Recommendation. *IEEE Transactions on Visualization and Computer Graphics*, 14(8):1–15, 2023. 2

[12] Y. Lin, H. Li, L. Yang, A. Wu, and H. Qu. InkSight: Leveraging Sketch Interaction for Documenting Chart Findings in Computational Notebooks. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):944 – 954, 2024. 2

[13] L. Liu. Pygwalker: Revolutionizing data analysis in jupyter notebooks. https://medium.com/@yuxuzi/pygwalker-revolutionizing-data-analysis-in-jupyter-notebooks-3c17eecb2000. [last accessed 16 June 2024]. 2

[14] K. Lo. *Optimisation de trafic de camions dans un contexte portuaire par une approche d'apprentissage machine*. PhD thesis, Université du Québec à Trois-Rivières, 2023. 4

[15] L. Y.-H. Lo, Y. Ming, and H. Qu. Learning vis tools: Teaching data visualization tutorials. In *2019 IEEE Visualization Conference (VIS)*, pp. 11–15, 2019. 4

[16] Y. Luo, X. Qin, N. Tang, and G. Li. Deepeye: towards automatic data visualization. In *Proceedings of the 34th IEEE International Conference on Data Engineering, ICDE'18*, pp. 101–112. IEEE, 2018. 2

[17] J. Mackinlay, P. Hanrahan, and C. Stolte. Show me: Automatic presentation for visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1137–1144, 2007. 3

[18] A. Mahalingappa. Global super store dataset. https://www.kaggle.com/datasets/apoorvaappz/global-super-store-dataset. [last accessed 16 June 2024]. 2

[19] A. McDonald. Pygwalker for exploratory data analysis in jupyter notebooks. https://www.youtube.com/watch?v=3WjWeH3HIMo&t. [last accessed 16 June 2024]. 2

[20] H. Mokhtarzadeh and S. Bagheri. Streamlining c3d file processing and visualization: A user-friendly approach using google colab and open-source python packages. 2023. 4

[21] C. North, R. Chang, A. Endert, W. Dou, R. May, B. Pike, and G. Fink. Analytic provenance: process+interaction+insight. In *Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, p. 33–36. ACM, 2011. 4

[22] OmdenaAI. Berlin-chapter-challenge-waste-management. https://github.com/OmdenaAI/Berlin-Chapter-Challenge-Waste-Management, 2023. [last accessed 16 June 2024]. 4

[23] OpenAI. Introducing chatgpt. https://openai.com/blog/chatgpt, 2022. [last accessed 16 June 2024]. 4

[24] B. Paget. Pygwalker: Simplifying exploratory data analysis with python. https://bryanpaget.medium.com/pygwalker-2f1de396df1f. [last accessed 16 June 2024]. 2

[25] A. Pandey, A. Srinivasan, and V. Setlur. MEDLEY: Intent-based Recommendations to Support Dashboard Composition. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):1135–1145, 2023. 2

[26] R. Polzin, S. Fritsch, K. Sharafutdinov, G. Marx, and A. Schuppert. Diagnostic expert advisor: A platform for developing machine learning models on medical time-series data. *SoftwareX*, 23:1–5, 2023. 4

[27] M. Raasveldt and H. Mühleisen. Duckdb: an embeddable analytical database. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, p. 1981–1984. ACM, 2019. 3

[28] A. Satyanarayan, B. Lee, D. Ren, J. Heer, J. Stasko, J. Thompson, M. Brehmer, and Z. Liu. Critical Reflections on Visualization Authoring Systems. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1–11, 2019. 2

[29] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2017. 3

[30] L. Shen, E. Shen, Y. Luo, X. Yang, X. Hu, X. Zhang, Z. Tai, and J. Wang. Towards Natural Language Interfaces for Data Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 29(6):3121–3144, 2023. 4

[31] L. Shen, E. Shen, Z. Tai, Y. Song, and J. Wang. TaskVis: Task-oriented Visualization Recommendation. In *Proceedings of the 23th Eurographics Conference on Visualization (Short Papers), EuroVis'21*, pp. 91–95. Eurographics, 2021. 2

[32] L. Shen, E. Shen, Z. Tai, Y. Xu, J. Dong, and J. Wang. Visual Data Analysis with Task-Based Recommendations. *Data Science and Engineering*, 7(4):354–369, 2022. 2

[33] L. Shen, Z. Tai, E. Shen, and J. Wang. Graph Exploration with Embedding-Guided Layouts. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–15, 2023. 4

[34] L. Shen, Y. Zhang, H. Zhang, and Y. Wang. Data Player: Automatic Generation of Data Videos with Narration-Animation Interplay. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):109–119, 2024. 4

[35] Streamlit. Streamlit — a faster way to build and share data apps. https://streamlit.io, 2023. [last accessed 16 June 2024]. 4

[36] S. R. Sukhdeve and S. S. Sukhdeve. Data visualization and business intelligence. In *Google Cloud Platform for Data Science: A Crash Course on Big Data, Machine Learning, and Data Analytics Services*, pp. 121–147. Springer, 2023. 4

[37] E. Wang. Bi-chatbot: Ai-powered bi. https://github.com/Ewen2015/BICat, 2023. [last accessed 16 June 2024]. 4

[38] Y. Wang, Z. Hou, L. Shen, T. Wu, J. Wang, H. Huang, H. Zhang, and D. Zhang. Towards Natural Language-Based Visualization Authoring. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):1222 – 1232, 2023. 4

[39] L. Wilkinson. *The Grammar of Graphics*. Springer, 2011. 3

[40] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):649–658, 2016. 2, 3

[41] A. Wu, Y. Wang, M. Zhou, X. He, H. Zhang, H. Qu, and D. Zhang. MultiVision: Designing Analytical Dashboards with Deep Learning Based Recommendation. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):162–172, 2022. 2

[42] Y. Wu, J. M. Hellerstein, and A. Satyanarayan. B2: Bridging Code and Interactive Visualization in Computational Notebooks. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology, UIST'20*, pp. 152–165. ACM, 2020. 2