A Data Driven Approach to Accessible Data Science

Venkatesh Potluri
vpotluri@cs.washington.edu

Sudheesh Singanamalla
sudheesh@cs.washington.edu

Nussara Tieanklin
nussara@cs.washington.edu

Abstract

Computational notebooks have become widely accepted and standardized tools for data science. these tools help data scientists to actively explore, analyze , and visualize the information, and finally convey the story of their data – all while enabling reproducibility, interactivity, and collaboration. Notebook tools like Jupyter, Datalore and Google Colab have actively been adopted for data science needs both in academia and industry and have been extended for use by various scientific communities. While there has been extensive research about how data scientists use computational notebooks, identify their pain points, and various attempts at enabling collaborative data science practices, very little is known about the various accessibility barriers experienced by blind or visually impaired (BVI) people using these notebooks. The visual nature of data representation and exploration prevents many blind and low vision or visually impaired (BVI) software engineers, and data scientists from being able to use the tools (1) due to the inaccessibility of the tools themselves, (2) due to the common ways in which data is represented through these tools, and (3) mechanisms to improve their accessibility. In this project proposal, we propose a data driven mechanism of analyzing over 10 Million Jupyter notebooks to identify accessibility challenges in published notebooks, understand their reproducibility, and identify/propose mechanisms to popular libraries & tools to improve the state of data science accessibility.

Introduction

Computational notebooks such as Jupyter, Google Colab, Datalore, and noteable (among others), are widely used by data scientists as interactive mechanisms to process, understand and express data making it easier for them to collaborate, share code, convey stories and narratives through data visualizations and text while keeping the reproducibility of results in mind [23, 35]. The popularity of these computational notebooks as the go-to tool for data science can be seen in the rapid increase in published notebooks, 2.5 Million public notebooks hosted on GitHub in September 2018 [23], increasing by 10x since 2015 [35]. More recently this dataset has increased to over 10 Million in 2020 when analyzed by JetBrains [7], a leading company building Integrated Development Environments (IDEs). The support for Jupyter (and similar) notebooks is actively being supported in many popular IDEs, such as VSCode, JetBrains, JupyterLab, Visual Studio, etc.., However, despite their popularity, Jupyter notebooks pose various challenges for data scientists and was systematically studied in the recent works by Chattopadhyay *et al.* [5]. Many issues such as challenges with deploying to production, notebook hosting, and long lived history and computation time requirements which were brought to light have recently been addressed in tools such as Google Colab, JetBrains Datalore, Databricks Notebooks and other hosted solutions.

Despite computational notebooks being popular tools, we know very little about the accessibility of these tools for developers and data scientists who are blind or visually impaired. For example,

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

As of writing this proposal – of the 253 citations of Rule *et al.*, the seminal work that presents a dataset of 2.5M notebooks [23], there is not even one mention of, or exploration of accessibility. Additionally, very little efforts have been made by the Jupyter and data science community to address or study the various accessibility challenges of data science tools thereby continuing the systemic inequity and posing barriers for entry for Blind and Visually Impaired (BVI) developers' engagement in data science. Integrations such as CodeTalk [18] and CodeWalk [20] in IDEs allow BVI developers to engage effectively with programming tasks. The capabilities of these integrations, however, do not extend to meet the very nuanced accessibility needs of BVI people interested in data science. The documentation for Jupyter – the most popular computational notebook platform – indicates that despite the overall knowledge of the inaccessibility since 2020, as of August 2022 during the community's last audit, the software continues to be completely inaccessible (grade F) for BVI developers with various challenges for colorblind users (grade C) [10]. This translates to BVI developers not being able to perform basic tasks such as navigate cells and read the output at various stages – which are very basic operations compared to the accessibility needs specific to datascience. As of writing this proposal, JupyterLab continues to be inaccessible and Jupyter notebook has not yet been analyzed with an in-depth accessibility review [34]. The use of other IDEs as a client to use Jupyter is not a viable alternative for BVI developers due to these other IDEs themselves not being accessible [15, 33]. This inherent inaccessibility of data science tools resulted in public requests by the BVI community to include accessible support for these tools [15].

Our work is a data-driven take to investigate the accessibility of datascience notebooks. we analyze a 10K randomly chosen subset from the 10 Million notebook dataset by JetBrains [7] as a need finding and systematic characterization exercise to identify various accessibility challenges in how notebooks are published by the data science community, analyze the types of visualizations, and identify target interventions for how the notebooks could be made more accessible. For this characterization, we:

1. Identify how data visualizations are published in Jupyter notebooks, and gain insights into what types of visualizations notebooks contain.
2. Understand the accessibility of Jupyter notebooks, and categorize accessibility issues introduced by popular themes and authoring practices by notebook users.

In summary, we hope to answer the following two overarching research questions through this work.

- ***RQ1****:* What accessibility challenges do blind or visually impaired users experience when using computational notebook software, and notebooks published using these tools?
- ***RQ2****:* What kind of target interventions are needed to make computational notebooks accessible to BVI consumers and notebook creators?

Through our work, we aim to contribute:

- A large scale analysis of the state of accessibility of information presented through computational notebooks.
- Targeted design interventions making computational notebooks accessible to screen reader users.
- Recommendations for notebook software makers, data scientists and accessibility researchers to make data and the corresponding story telling accessible.

Background

Several efforts have attempted to understand and address the access barriers experienced by BVI software developers. Mealin and Murphy-Hill [14] published one of the first studies to understand accessibility barriers experienced by BVI developers. They present challenges associated with using IDEs and developing user interfaces. Many followup studies [1, 24, 32] highlight accessibility challenges BVI developers face in seeking information critical to software development, and with specific tasks and tools such as code navigation and command line interfaces. Researchers built tools to address specific accessibility barriers posed by software engineering practices and evaluated their impact on BVI developers. for example, several tools [3, 6, 17, 25] present novel, accessible representations of code that facilitate efficient screen reader navigation. Similarly, researchers contributed audio-based techniques to facilitate accessible debugging of code [17, 30, 31]. These code navigation and debugging tools address accessibility barriers encountered when using integrated development environments (IDEs) – which are actively used in the data science community through tools like JupyterHub. While these tools address accessibility barriers for BVI developers to perform some tasks, they are not sufficient for BVI developers to accessibly perform programming tasks involving expert domains.

Web and User Interface/Experience development is one such domain that recently received attention from accessibility researchers. Kearney-Volpe *et al.* [11] highlight various challenges with making web development accessible. The authors specifically note various barriers experienced by BVI developers in making decisions related to styling and building the interface for web pages – a programming domain extensively investigated and iteratively addressing various accessibility challenges [12, 13, 16, 19, 28]. The solutions proposed through the investigations range from improvements to IDEs, additional/custom software tools, and physical tactile interfaces.

A critical aspect of using computational notebooks is to create, consume, and collaborate on visual representations of data. several efforts have explored making data visualizations accessible through auditory representations i.e. combining speech and tones [8, 26], making these auditory representations interactive through voice commands [27], in addition to touch screen interactions and keyboard shortcuts [2, 27]. These efforts assume BVI people as non-expert consumers of data visualizations. Computational notebooks however, in addition to enabling consumption of data visualizations, give BVI developers the means to produce data visualizations and enable data driven story telling, therefore surfacing the need for these tools to offer capabilities to provide accessible representations of these data visualizations. Very few efforts have resulted in tools and interfaces for BVI people to author accessible data visualizations. As of today, the work by Cantrell *et al.* resulting in the development of Highcharts Sonification Studio is the only open source charting library to support accessible authoring of data sonifications [4]. Potluri *et al.* [21] developed a data sonification toolkit, centered around the needs of BVI developers' attempts at and need for understanding sensor data and enable them to develop Internet of Things (IoT) applications.

Though these tools and efforts make data accessible to BVI developers and users, very little is known about the applicability of these efforts when using computational notebooks.

Analysis of Jupyter Notebooks

To examine the accessibility of datascience notebooks, we begin by analyzing the JetBrains ten million notebooks dataset [7]. Our main foci were to:

1. Setup a scalable pipeline to clean, and process the data to identify supporting data driven evidence for accessibility.
2. Apply chart classification approaches to identify the different types of plots.
3. Develop an initial understanding of the different types of plots and plotting libraries used in these notebooks to facilitate further analysis.
4. perform accessibility analysis on HTML exports of notebooks and compare accessibility of notebook content and accessibility across different themes.

To focus on these investigations, and to prepare our data processing pipelines before applying it to the large dataset, we first chose a random sample of 1000 notebooks from the 10M notebook dataset and reported some findings for our milestone deliverable. **In this final deliverable, we include additional findings on a random sample of 10000 notebooks, and continue to prepare our setup for further large scale analysis – whose results we hope to submit for publication to the upcoming ASSETS conference.**

Data Preprocessing

To answer our research questions, we analyzed code, output, and markdown cells of Jupyter notebooks, and performed automated accessibility audits on html renders of notebooks. This analysis required notebooks to be processed both in their raw iPython format, and as HTML exports.

Our analysis pipelines first filtered notebooks that were valid (correctly formatted) `ipynb` Jupyter notebook files (identified using the `nbformat` library), and extracted the different types of image outputs and stored them as base64 strings. To identify the types of charts in these images, we converted them into `Image` instances and classified them using the pre-trained model released by Jobin *et al.* [9]. To gain further insight into the type of libraries notebook users used to visualize data, we filtered for notebooks written in python (both v2 and v3), that had at least 1 cell with a visual representation of data. We performed code analysis on the code cells of these notebooks to obtain information about the libraries that notebook authors used to visualize data, and on the presence of subfigures. To gain insights into the accessibility of popular themes resembling IDEs that developers use to author Jupyter notebooks, we exported the notebooks as HTML using `nbconvert` – the default used by Jupyter's export to HTML format. We exported notebooks to html using the *solarized* theme (originally written for Vim and made available now by various IDEs), *Darkula* (for ZSH originally, and used extensively by JetBrains), *Horizon* (default by VSCode), the *Material Darker* theme, and Jupyter's default *dark* and *light* themes.

We perform accessibility scans by deploying a self hosted version of the *Pa11y* accessibility scanning infrastructure, and configuring the services to using both the `aXe` and `htmlcs` runners.

We ran all our analysis on a AWS p3.2xlarge machine with 8 CPU cores, a 16 GB NVidia Tesla V100 GPU and 64GB of RAM. For larger scale analysis we plan to run in the future, we hope to scale our experimental setup to multiple machines managing spark and an underlying Hadoop file system or using the notebooks currently stored in AWS S3 document storage. As of this report, we could only scale to 10K notebooks on the single VM because of cost reasons. The pa11y infrastructure runs 16 concurrent Chromium tasks for scanning the accessibility of each exported notebook and incurs a significant compute overhead since each executions spins up and destroys a sandboxed chromimum process. We hope to address some of these scalability challenges in future iterations as we continue to work towards a publication.

1 Results

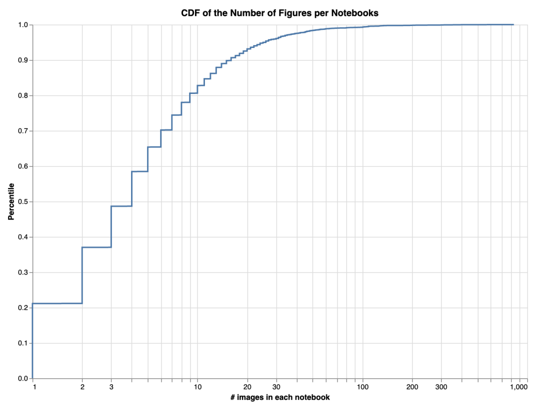
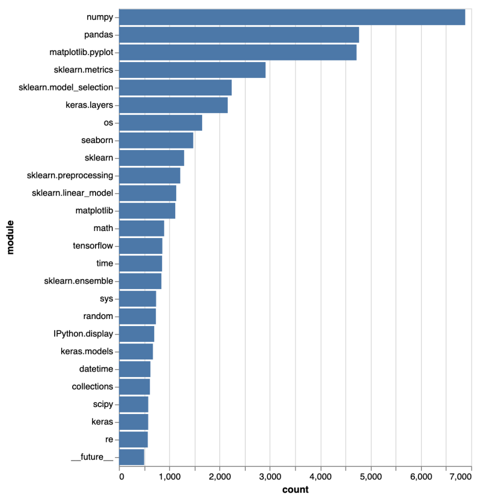


Figure 1: Distribution of the number of plots contains in each notebook.

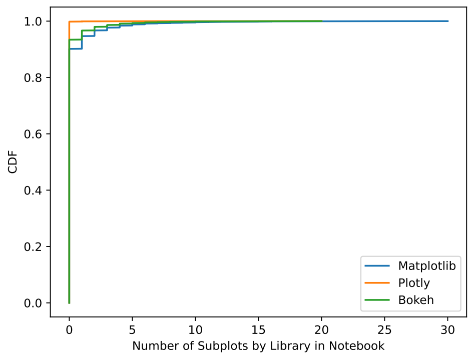
Our results show that, of the 10000 randomly chosen notebooks, we were able to successfully parse 9949 notebooks from the Amazon S3 bucket provided by the authors of the dataset, since 51 others were corrupted. Of these, we identified that 9676 files (97.25%) had information about the language of the code used in the notebook. 94.99% of these notebooks were written in python due to it's popularity in data science across people with various levels of expertise. The notebooks we consider for this report contained a total of 33353 figures. Of the 9949 notebooks, we were able to convert 9836 notebooks into HTML using `nbconvert` and categorized them by theme, resulting in 59016 exported Jupyter notebooks.

1.1 Characterization: Visual Data Representations

In Figure 1 we present a cumulative distribution of the number of figures in each notebook. At the median, a notebook contains 4 or more figures with the 90th percentile indicating over 15 figures in a notebook. There exists a long tail of notebooks which contain over 100 figures. We did not perform an alt-text analysis (*i.e.,* if images have alt text) as we are aware that there is currently no way for notebook authors to add these descriptions to plots that are generated in programmatic ways. However, the prevalence of figures with no ability to enable alt text, to convey data in notebooks calls for targeted efforts to make them accessible. We focused our analysis to uncover information to help guide some of our initial explorations and future efforts to make data visualizations accessible in the context of data science and notebooks.



(a) Imported modules Ranked by Usage Frequency



(b) CDF of Number of Subplots in Figures

Figure 4

We classified these images by the type of visualization using the DocFigure pre-trained model. We found that line chart was the most common type of visualization that notebook users plotted, followed by box plots and histograms. Figure 2 contains a histogram of the most popular types of figures found in the analyzed notebooks. In Figure 3 we analyze the type of output format of the images in the Jupter notebooks and identify that png (n=32512) is disproportionately the most widely used graphical output format used by notebook authors, followed by svg (n=255), and jpg (n=80).

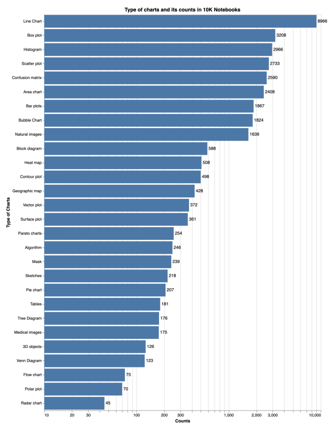


Figure 2: The visualization types used in the random 10000 notebooks.

We also identify the different types of modules that notebook users imported to find the most frequently used libraries to generate plots in the figure. We did this by implementing a parser in our pipeline which uses an abstract syntax tree `ast` and extracts a named tuple from the code matching the module and function imports using the `ast.Import`, and `ast.ImportFrom`. Figure 4a shows a bar chart of these imports. Matplotlib, Seaborn and IPython display were the most popular visualization libraries used, with other libraries such as Plotly, and Bokeh being used within the top 100 most popular python libraries for visualizations. Identifying the set of imports through code analysis is particularly challenging because of the various simplification mechanisms provided by Jupyter notebooks similar to macros eg. `%matplotlib inline`. While these lines of code starting with % are valid within environments that render notebooks, they are not a part of the programming language syntax and result in errors when the AST is used as-is. This also includes command sequences in notebook starting with `!` and documentation help features presented by the special character `?`.

The information obtained from the above process helps us identify the popular libraries which are used by data scientists to generate graphical representations of their data. Scientific papers frequently use subfigures to report information related to experimental results together [29]. We hypothesized the generation of the same in Jupyter notebooks during authoring and identified the number of figures being generated as subfigures. To do this, we chose three popular graphing modules `matplotlib`, `plotly`, and `boken`, and implement syntax parsers to understand more about the characteristics of the plots being rendered. We extract the number of subfigures by identifying the number of function calls to `GridPlot`, `facet`, `column`, `layout`, or `subplots` and present our results in Figure 4b. Our results indicate that ≈90% of figures in our analysis do not have any subfigures, 2% images have 1 subfigure (2 figures), and the longer tail creating over 5 subfigures. We find that data scientists using `plotly` typically (≥99% of the time) generate only one image with no subfigures, revealing that the authoring practices from LaTeX does not translate directly to the way in which graphics are generated in Jupyter notebooks.

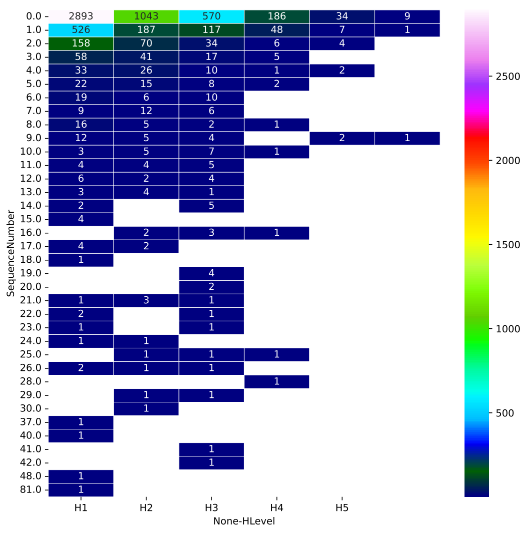


Figure 5: First Interactive Element

1.2 Identifying First Navigable Elements

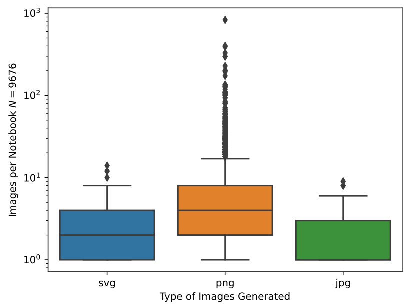


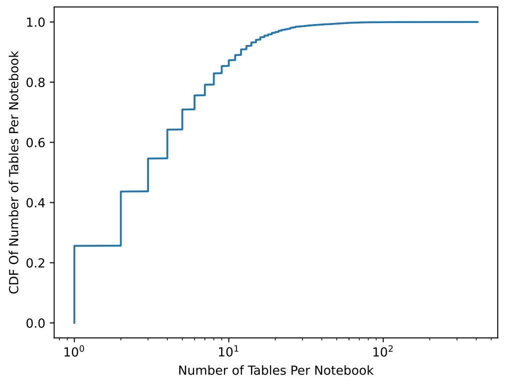
Figure 3: Type of Figures Included in Jupyter Notebooks

Screen reader users rely on basic keyboard commands for reading and navigating web content, a basic and most frequently used method of interaction when dealing with web based interactions is to jump to the first heading level, typically described by `<H1>` level among the six available in the HTML web standards (`<H1...6>`). To understand, authoring practices, we analyze the notebooks for where their *first* interactive heading element lies in the entire notebook and present our results in the Heatmap shown in Figure 5. Screen reader users often look for a heading level 1 to gain context of a notebook. They often do so by pressing the “1” (to jump to the first heading level 1) or “h” (to jump to the first heading) key after a webpage load. We find a majority of notebooks, have a heading level 1 in the first cell (cell 0 in Figure 5). We speculate Jupyter's default behavior of adding a heading level 1 in the first cell when a notebook is created may have resulted in this accessibility advantage. While most notebooks have the highest heading level in the first cell of the notebook, over 15% of them have a lower heading level in the first cell. $\geq$7% of notebooks have H1 as their first interactive element but is only made available after the first cell and in some extreme cases not available even until the 10th cell. This incredibly complicates the navigational experiences resulting in a BVI data scientist skipping many relevant cells of the notebooks. The higher number of *holes* in the matrix corresponding to H2 and H3 levels indicate more data scientists skipping navigational headers structures, possibly for aesthetic reasons exposed by CSS style sheets.

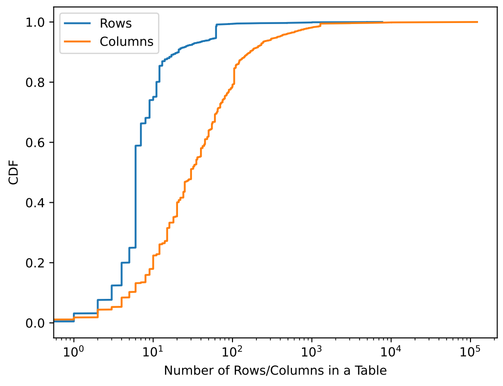
The presence of a heading in the first cell however does not guarantee that it is used in the most effective way to provide a title to the user, and additional analysis (out of scope for this report) is required to assess the relevance of these headings.

1.3 Tables and Interactivity

When considering the skimmability of a notebook, we also accounted for tables to be skimmable elements. Screen reader users can jump to the first table by pressing the “t” key. Tables are currently



(a) CDF Of Number of Tables Per Notebook



(b) CDF Of Number of Rows and Columns in a Table

Figure 6: Tables and Interactivity

the most accessible way of representing data to a screen reader user. The presence of tables increases the possibility for a screen reader user to quickly jump to results if the intent for looking into a notebook is to quickly gain some insights. While the presence of links and graphics could also be considered as the presence of skimmable elements (due to screen readers having capabilities to navigate to them by pressing a single key), we do not account for them as they can be too specific (in case of links) and be inaccessible (in the case of graphics) to screen reader users. While there are 6351 notebooks which contain the first navigable element as a heading, 1143 of the notebooks have tables as their first navigable element, while 39 notebooks have both a heading and a table made available for navigation in the same cell.

We present the distribution of the number of tables per notebook and identify that 20% of the notebooks have no tables in them, and at the median, notebooks contain 3 tables, while at the 90th percentile, they contain 12 tables in a notebook as shown in Figure 6a. However, having a table by itself doesn't make the notebooks accessible since a lot of rows and columns can affect screen reader navigation often resulting in users losing context or requiring too many key strokes to skip the tables or interact with elements in the table. By analysing the typical number of rows and columns present in these tables, we observe that a median table presents 6 rows of data in a Jupyter output but contains over 30 columns. The row count is explained by the default behavior of data frame display functions such as `pandas.head()` which are invoked when data is rendered. The number of columns however expand heavily with tables in the 90th percentile (10% of notebooks) containing over 168 columns in their tables as shown in Figure 6b.

1.4 Accessibility Effects of Themes

The themes used in IDEs and webpages affect accessibility especially for those with low vision and visual impairments. Applying accessible color scheme such as high contrast themes on IDEs has been widely adopted by these developers for both accessibility and aesthetic reasons. These different color combinations can help highlight syntax, personal customization, and improve accessibility. However, Jupyter notebooks exported to HTML will always be presented in the default *light* theme regardless of the developers' IDE theme. Though this standardization might be helpful in how users interact with notebooks, the default results in notebooks being circulated in formats which might be inaccessible. We run an accessibility tool *pa11y* with two open source accessibility engines aXe and HTMLCS, and present the distribution of the number of accessibility warnings and errors in Figure 7. Our analysis shows that for the same set of notebooks, the light theme generated performs considerable worse in terms of accessibility and generates a heavy number of color contrast based accessibility issues. A simple change, such as exporting the notebooks using the *horizon* theme which is the default view for popular IDEs such as VSCode, we could significantly reduce the accessibility errors from 188 median errors to 26 indicating a 7.2x improvement in accessibility. The darcula theme performs the worst with most number of errors (204 at median), while the alternative to the default (dark theme) is 1.4x better than light theme. Therefore, we suggest Jupyter notebook engineers to provide theme picker capabilities in the rendered HTML of the notebooks while defaulting to a more accessible theme.

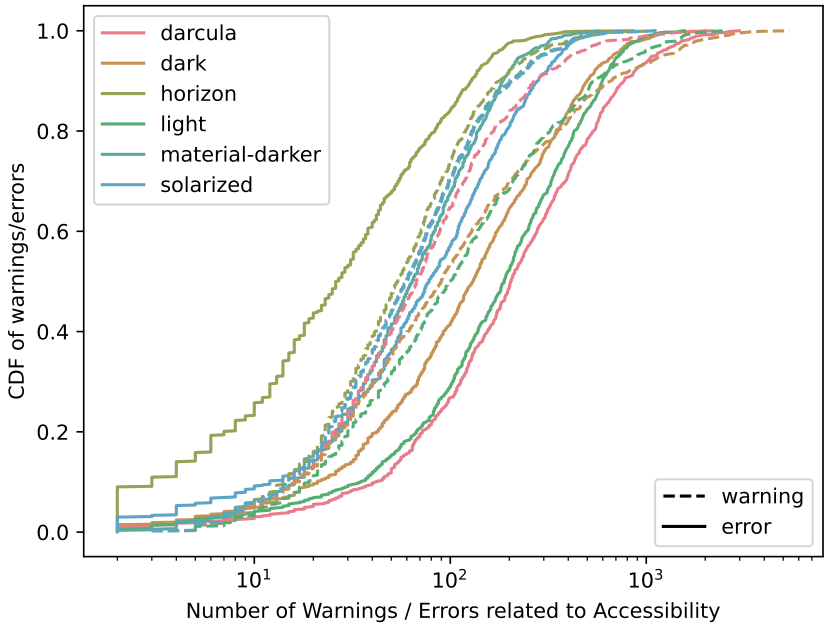


Figure 7: Number of Accessibility Warnings and Errors Based on Jupyter Theme

1.5 Changes to Tooling

Based on the results obtained, we chose a popular python module `matplotlib`, and the png output format based on its popularity and introduced the ability for a developer to configure alternate text in their figures when the `show()` or the `savefig()` functions are invoked. To enable this, we expose the ability pass a *metadata* dictionary with the `alt` keyword in the function arguments. An example usage can be seen below:

```python
import matplotlib.pyplot as plt

for file_extension in ["png"]:
    fig, ax = plt.subplots(nrows=1, ncols=1)
    X = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    Y = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    ax.plot(X, Y, label=rf"$X=Y$ Line")
    plt.savefig(f"dummy_image.{file_extension}",
                bbox_inches="tight",
                metadata={
                    "alt": f"This is an example description of the ALT
     text for the {file_extension} image"
                })
```

We update the `matplotlib` target backend for `PNG` formats, and modify the matplotlib source code (`lib/matplotlib/image.py`, `lib/matplotlib/backends/backend_(png|pdf).py`), to use the Exchangeable Image File Format (`EXIF`) and include alt text information. The Image description information is encoded in a serialized byte format against the `0x010e (270)` byte delimiter following EXIF standards allowing HTML and other readers to programmatically read information. This information can be obtained by the Jupyter notebook and encoded into the resulting encoded base64 image in the notebook files. We believe, this interface is one of *many* ways in which data scientists could be provided pro-active mechanisms to ensure the accessibility of their notebooks.

2 Future Work and Challenges

During the course of this project, We were able to develop an understanding of our dataset, and set up a scalable working pipeline to scale our analysis to more notebooks, while helping us narrow down on a suitable chart classification technique, and perform some initial code analysis. While we cannot fully detail out all the results we have in this report due to space constraints, our work towards a publication additionally focuses on: (1) more extensive code analysis approaches (color and usage identification), (2) detail and classify the types of common accessibility errors from large scale web accessibility scans, (3) image processing techniques to identify the types of colors used in a figure and their accessibility (color contrast for visual disabilities such as dueteranomaly, proanomaly, protanopia, and deuteranopia among many others), (4) identify visual crowding in images using Feature Congestion and Subband Entropy mechanisms [22], (5) changes to the tooling for `nbconvert` to support theme-pickers and accessible defaults.

The path to this progress had more hurdles than we anticipated. Broadly, these can be bucketed as caused by (1) incompleteness of preprocessed data provided by the JetBrains team, (2) incorrect notebook cell ordering by users resulting in default parsers considering the code samples as syntax errors (3) limitations related to accessing computational resources to perform larger scale data analysis and chart classification.

While the focus of the work is not reproducibility of the notebook itself, incorrect execution order could have implications around accessibility for BVI data scientists attempting to re-execute these notebooks as python scripts (due to their inherent accessibility).

Incomplete pre-processed data Though JetBrains provided us with a notebook and some associated data files that helped them with their analysis of 10M notebooks, this pre-processed data did not prove to be useful in a way that we could extend their findings without needing to extensively reanalyze for different information. Though their analysis provides information about the languages of notebooks, and imports in them, the data does not contain any information to the associated code file for each entry. This required that we re-create steps to identify the different languages used in these notebooks, and the relevant imports.

Cell ordering inconsistencies Jupyter notebooks maintain the state when a cell is executed allowing users to execute cells in any order, as long as the execution steps are syntactically correct. Code analysis however assumes that code is executed in a sequential order as it appears, resulting in an invalid syntax when it is converted into an AST. While we had to drop one notebook due to an inconsistency of this nature, we anticipate this to be a recurring issue that we will not be able to account for.

Limitations in Computing Resources As indicated in our proposal, the size of the dataset is relatively large, requiring us to have access to high capacity storage. Additionally, the chart classification model requires an NVIDIA GPU with Cuda support to classify images with a reasonable speed.

An additional challenge we encountered was to understand the intent of notebooks i.e. if a notebook is a tutorial, research artifact, or intended for self exploration. This information could have helped us gain deeper insights into where the target interventions are required for improving accessibility. For example, if the notebooks with most accessibility issues are tutorials or homeworks, it implies a high barrier to entry to BVI data scientists to learn the necessary skills and toolsets. Similarly, these errors existing on research artifacts prevents BVI developers from easily building on top of existing research which is common practice in the machine learning community.

We are excited to continue our analysis and address the challenges ahead, scale the measurements towards 100K and look forward to submitting our results as a full publication.

References

[1] Khaled Albusays, Stephanie Ludi, and Matt Huenerfauth. Interviews and observation of blind software developers at work to understand code navigation challenges. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '17, page 91–100, New York, NY, USA, 2017. Association for Comput-

ing Machinery. ISBN 9781450349260. doi: 10.1145/3132525.3132550. URL `https://doi.org/10.1145/3132525.3132550`.

[2] Apple. Apple developer documentation: Audio graphs, 2022. `https://developer.apple.com/documentation/accessibility/audio_graphs`.

[3] Catherine M. Baker, Lauren R. Milne, and Richard E. Ladner. Structjumper: A tool to help blind programmers navigate and understand the structure of code. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, page 3043–3052, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450331456. doi: 10.1145/2702123.2702589. URL `https://doi.org/10.1145/2702123.2702589`.

[4] Stanley J Cantrell, Bruce N Walker, and Øystein Moseng. Highcharts sonification studio: an online, open-source, extensible, and accessible data sonification tool. In *26th International Conference on Auditory Display (ICAD'21)*. Georgia Institute of Technology, 2021.

[5] Souti Chattopadhyay, Ishita Prasad, Austin Z Henley, Anita Sarma, and Titus Barik. What's wrong with computational notebooks? pain points, needs, and design opportunities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2020.

[6] Md Ehtesham-Ul-Haque, Syed Mostofa Monsur, and Syed Masum Billah. Grid-coding: An accessible, efficient, and structured coding paradigm for blind and low-vision programmers. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393201. doi: 10.1145/3526113.3545620. URL `https://doi.org/10.1145/3526113.3545620`.

[7] Alena Guzharina. We downloaded 10,000,000 jupyter notebooks from github – this is what we learned | the jetbrains datalore blog. `https://blog.jetbrains.com/datalore/2020/12/17/we-downloaded-10-000-000-jupyter-notebooks-from-github-this-is-what-we-learned/`, 12 2020. (Accessed on 01/18/2023).

[8] Leona Holloway, Cagatay Goncu, Alon Ilsar, Matthew Butler, and Kim Marriott. Infosonics: Accessible infographics for people who are blind using sonification and voice. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022. doi: 10.1145/3491102.3517465.

[9] K. V. Jobin, Ajoy Mondal, and C. V. Jawahar. Docfigure: A dataset for scientific document figure classification. In *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*, volume 1, pages 74–79, 2019. doi: 10.1109/ICDARW.2019.00018.

[10] Man From Jupyter. Accessibility issues needing addressing for wcag 2.1 compliance (as of version 2.2.6) · issue #9399 · jupyterlab/jupyterlab. `https://github.com/jupyterlab/jupyterlab/issues/9399`, 11 2020. (Accessed on 01/18/2023).

[11] Claire Kearney-Volpe and Amy Hurst. Accessible web development: Opportunities to improve the education and practice of web development with a screen reader. *ACM Trans. Access. Comput.*, 14(2), July 2021. ISSN 1936-7228. doi: 10.1145/3458024. URL `https://doi.org/10.1145/3458024`.

[12] Jiasheng Li, Zeyu Yan, Ebrima Haddy Jarjue, Ashrith Shetty, and Huaishu Peng. Tangiblegrid: Tangible web layout design for blind users. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393201. doi: 10.1145/3526113.3545627. URL `https://doi.org/10.1145/3526113.3545627`.

[13] Jingyi Li, Son Kim, Joshua A. Miele, Maneesh Agrawala, and Sean Follmer. Editing spatial layouts through tactile templates for people with visual impairments. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pages 206:1–206:11, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-5970-2. doi: 10.1145/3290605.3300436. URL `http://doi.acm.org/10.1145/3290605.3300436`.

[14] Sean Mealin and Emerson Murphy-Hill. An exploratory study of blind software developers. In *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 71–74, Innsbruck, Austria, 2012. IEEE. doi: 10.1109/VLHCC.2012.6344485.

[15] mltony. Feature request: Accessibility support for jupyter notebooks in vscode · issue #90408 · microsoft/vscode. `https://github.com/microsoft/vscode/issues/90408`, 02 2020. (Accessed on 01/18/2023).

[16] Maulishree Pandey, Sharvari Bondre, Sile O'Modhrain, and Steve Oney. Accessibility of ui frameworks and libraries for programmers with visual impairments. In *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–10, 2022. doi: 10.1109/VL/HCC53370.2022.9833098.

[17] Venkatesh Potluri, Priyan Vaithilingam, Suresh Iyengar, Y. Vidya, Manohar Swaminathan, and Gopal Srinivasa. Codetalk: Improving programming environment accessibility for visually impaired developers. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–11, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356206. doi: 10.1145/3173574.3174192. URL `https://doi.org/10.1145/3173574.3174192`.

[18] Venkatesh Potluri, Priyan Vaithilingam, Suresh Iyengar, Y. Vidya, Manohar Swaminathan, and Gopal Srinivasa. Codetalk: Improving programming environment accessibility for visually impaired developers. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–11, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356206. doi: 10.1145/3173574.3174192. URL `https://doi.org/10.1145/3173574.3174192`.

[19] Venkatesh Potluri, Liang He, Christine Chen, Jon E. Froehlich, and Jennifer Mankoff. A multi-modal approach for blind and visually impaired developers to edit webpage designs. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '19, page 612–614, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366762. doi: 10.1145/3308561.3354626. URL `https://doi.org/10.1145/3308561.3354626`.

[20] Venkatesh Potluri, Maulishree Pandey, Andrew Begel, Michael Barnett, and Scott Reitherman. Codewalk: Facilitating shared awareness in mixed-ability collaborative software development. In *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392587. doi: 10.1145/3517428.3544812. URL `https://doi.org/10.1145/3517428.3544812`.

[21] Venkatesh Potluri, John Thompson, James Devine, Bongshin Lee, Nora Morsi, Peli De Halleux, Steve Hodges, and Jennifer Mankoff. Psst: Enabling blind or visually impaired developers to author sonifications of streaming sensor data. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393201. doi: 10.1145/3526113.3545700. URL `https://doi.org/10.1145/3526113.3545700`.

[22] Ruth Rosenholtz, Yuanzhen Li, Jonathan Mansfield, and Zhenlan Jin. Feature congestion: a measure of display clutter. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 761–770, 2005.

[23] Adam Rule, Aurélien Tabard, and James D Hollan. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.

[24] Harini Sampath, Alice Merrick, and Andrew MacVean. *Accessibility of Command Line Interfaces*, pages 1–10. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450380966. URL `https://doi.org/10.1145/3411764.3445544`.

[25] Emmanuel Schanzer, Sina Bahram, and Shriram Krishnamurthi. Accessible ast-based programming for visually-impaired programmers. In *Proceedings of the 50th ACM Technical Symposium*

on Computer Science Education, SIGCSE '19, page 773–779, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450358903. doi: 10.1145/3287324.3287499. URL `https://doi.org/10.1145/3287324.3287499`.

[26] Ather Sharif, Sanjana Shivani Chintalapati, Jacob O. Wobbrock, and Katharina Reinecke. Understanding screen-reader users' experiences with online data visualizations. In Jonathan Lazar, Jinjuan Heidi Feng, and Faustina Hwang, editors, *ASSETS '21: The 23rd International ACM SIGACCESS Conference on Computers and Accessibility, Virtual Event, USA, October 18-22, 2021*, pages 14:1–14:16. ACM, 2021. doi: 10.1145/3441852.3471202. URL `https://doi.org/10.1145/3441852.3471202`.

[27] Ather Sharif, Olivia H. Wang, Alida T. Muongchan, Katharina Reinecke, and Jacob O. Wobbrock. Voxlens: Making online data visualizations accessible with an interactive javascript plug-in. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391573. doi: 10.1145/3491102.3517431. URL `https://doi.org/10.1145/3491102.3517431`.

[28] Ashrith Shetty, Ebrima Jarjue, and Huaishu Peng. Tangible web layout design for blind and visually impaired people: An initial investigation. In *Adjunct Publication of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST '20 Adjunct, page 37–39, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450375153. doi: 10.1145/3379350.3416178. URL `https://doi.org/10.1145/3379350.3416178`.

[29] Noah Siegel, Zachary Horvitz, Roie Levin, Santosh Divvala, and Ali Farhadi. Figureseer: Parsing result-figures in research papers. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VII 14*, pages 664–680. Springer, 2016.

[30] Andreas Stefik, Roger Alexander, Robert Patterson, and Jonathan Brown. Wad: A feasibility study using the wicked audio debugger. In *15th IEEE International Conference on Program Comprehension (ICPC '07)*, pages 69–80, 2007. doi: 10.1109/ICPC.2007.42.

[31] Andreas Stefik, Andrew Haywood, Shahzada Mansoor, Brock Dunda, and Daniel Garcia. Sodbeans. In *2009 IEEE 17th International Conference on Program Comprehension*, pages 293–294, Vancouver, BC, Canada, 2009. IEEE.

[32] Kevin M. Storer, Harini Sampath, and M. Alice Merrick. "it's just everything outside of the ide that's the problem": Information seeking by software developers with visual impairments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, May 2021. Association for Computing Machinery. ISBN 9781450380966. doi: 10.1145/3411764.3445090. URL `https://doi.org/10.1145/3411764.3445090`.

[33] Sublime users. A request for the implementation of accessibility. issue #3392. sublimehq/sublime_text, 2020. URL `https://github.com/sublimehq/sublime_text/issues/3392`.

[34] Jupyter Accessibility Team. Jupyter accessibility — jupyter accessibility - team compass. `https://jupyter-accessibility.readthedocs.io/en/latest/`, 08 2022. (Accessed on 01/18/2023).

[35] Jiawei Wang, Li Li, and Andreas Zeller. Better code, better sharing: on the need of analyzing jupyter notebooks. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, pages 53–56, 2020.