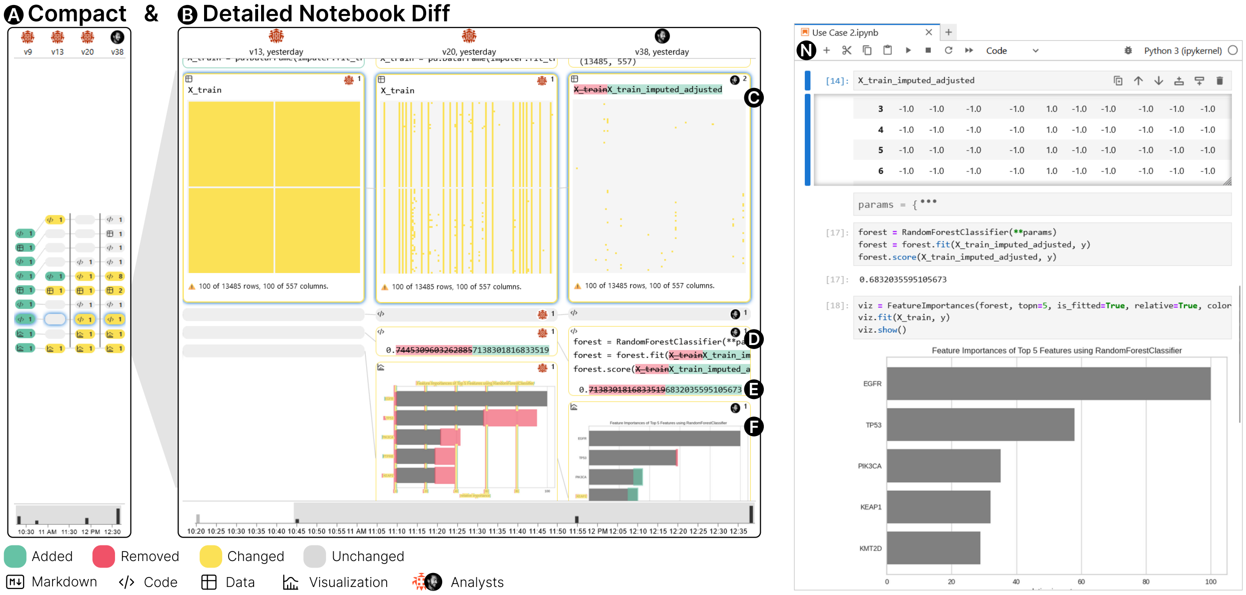
Loops: Leveraging Provenance and Visualization to Support Exploratory Data Analysis in Notebooks

Klaus Eckelt, Kiran Gadhave, Alexander Lex, and Marc Streit



```
X_train
100 of 13485 rows, 100 of 557 columns.
(13485, 557)
~~X_train~~X_train_imputed_adjusted
0.~~7445309603262885~~7138301816833519
forest = RandomForestClassifier(**pa
forest = forest.fit(~~X_train~~X_train_im
forest.score(~~X_train~~X_train_imputed_a
0.~~7138301816833519~~6832035595105673
```

```
[14]: X_train_imputed_adjusted
   3  -1.0  -1.0  -1.0  -1.0  1.0  -1.0  -1.0  -1.0  -1.0  -1.0
   4  -1.0  -1.0  -1.0  -1.0  1.0  -1.0  -1.0  -1.0  -1.0  -1.0
   5  -1.0  -1.0  -1.0  -1.0  1.0  -1.0  -1.0  -1.0  -1.0  -1.0
   6  -1.0  -1.0  -1.0  -1.0  1.0  -1.0  -1.0  -1.0  -1.0  -1.0

      params = {•••

[17]: forest = RandomForestClassifier(**params)
      forest = forest.fit(X_train_imputed_adjusted, y)
      forest.score(X_train_imputed_adjusted, y)

[17]: 0.6832035595105673

[18]: viz = FeatureImportances(forest, topn=5, is_fitted=True, relative=True, color
      viz.fit(X_train, y)
      viz.show()
```

Fig. 1: A Loops enabled notebook showing the evolution of the analysis. While the notebook N only shows the most recent state,Loops visualizes the notebook history next to the notebook. The compact notebook diffs A provide an overview of how the notebook's structure and content changed over time. The rectangles represent the notebook cells, color-coded according to their status relative to the previous version. We show the type of cell (markdown, code, tables, visualizations/images) using an icon and show the execution count where appropriate. The detailed notebook diffs B reveal how cell content changed, using difference visualizations specific to the type of content; including tables C, code D, text E, and visualizations F.

Abstract—Exploratory data science is an iterative process of obtaining, cleaning, profiling, analyzing, and interpreting data. This cyclical way of working creates challenges within the linear structure of computational notebooks, leading to issues with code quality, recall, and reproducibility. To remedy this, we present Loops, a set of visual support techniques for iterative and exploratory data analysis in computational notebooks. Loops leverages provenance information to visualize the impact of changes made within a notebook. In visualizations of the notebook provenance, we trace the evolution of the notebook over time and highlight differences between versions. Loops visualizes the provenance of code, markdown, tables, visualizations, and images and their respective differences. Analysts can explore these differences in detail in a separate view. Loops not only improves the reproducibility of notebooks but also supports analysts in their data science work by showing the effects of changes and facilitating comparison of multiple versions. We demonstrate our approach's utility and potential impact in two use cases and feedback from notebook users from various backgrounds. This paper and all supplemental materials are available at https://osf.io/79eyn.

Index Terms—Comparative visualization, computational notebooks, provenance, data science

1 Introduction

Computational notebooks are the tool of choice in many data science applications [10]. As a literate programming tool [35], notebooks combine analysis and narration for the analysis in a single document. Notebooks can integrate diverse formats, such as different programming languages, multimedia content, interactive widgets, and visualizations as part of the output. The most commonly used notebook technology is the Jupyter family [34], the use of which has doubled annually in recent years [51].

The promise of literate programming—interspersing explanatory text with code—is reproducibility of the result. However, previous research has shown that reproducibility is not the norm when using computational notebooks [22, 49, 64, 67]. A large-scale study by Pimentel et al. [49] showed that only 24.11% of 863,878 public Jupyter Notebooks could be re-executed and just 4.03% produced the same results. Computational notebooks "can foster poor coding practices" [47], become messy [23], and as a result, are often not reproducible [22, 49]. Poor coding and documentation practices also affect data analysis reproducibility, as analysts often start with a vague understanding of their resources and goals [3, 25].

The information foraging and sensemaking loop describes how analysts search, filter, and extract information, continuously developing a mental model that aligns with the new information and their existing knowledge [50]. In the series of loops that make up the analysis process, each iteration refines the understanding and brings the analyst closer to

- *Klaus Eckelt and Marc Streit are with Johannes Kepler University Linz, Austria. E-mail: klaus@eckelt.info; marc.streit@jku.at.*
- *Kiran Gadhave and Alexander Lex are with University of Utah, United States. E-mail: kirangadhave2@gmail.com; alex@sci.utah.edu.*

their goal. But an iterative analysis process does not match well with the notebook's linear structure; hence, it can be difficult for analysts to understand the state of their notebook. For example, analysts often update their code to reflect new insights or hypotheses, losing track of previous attempts and results [54]. Alternatively, they may duplicate their code to compare different approaches, resulting in messy notebooks that contain outdated or redundant code [54, 66]. Another issue, specific to sequentially executed notebooks such as Jupyter, are out-of-order executions that break the logical flow and introduce hidden dependencies [23, 49, 54]. These practices make reproducing the analysis process and results difficult for analysts and others who want to reuse or verify their work.

Previous works tried to mitigate these issues by capturing the provenance of notebooks [31, 55] or providing live support for exploratory and/or iterative data science tasks [14, 15]. However, to date, these two approaches have been studied independently and rarely combined.

To remedy this, we introduce Loops, our **primary contribution**, a novel visual exploration approach that tracks and visualizes the changes of a notebook over time, allowing analysts to better understand the impact of their changes and the notebook's history. Loops juxtaposes the notebook with compact representations of the analysis "loops". Loops visualizes changes made to the notebook and how they impact the output. By tracking the provenance of notebooks and visualizing the history and differences, we can assist analysts in their ongoing analysis by revealing the impact of their changes. We also contribute a discussion of the different types of content found in computational notebooks and how differences in these contents between notebook versions can be effectively visualized. We implement Loops as an open-source JupyterLab extension, available at `https://github.com/jku-vds-lab/loops`. We validate our approach through use cases and feedback from notebook users that demonstrate its utility.

2 Related Work

We start this section by discussing research that studied the prevalent issues of computational notebooks and the challenges analysts face when working with them. We then discuss how the provenance of computational notebooks can be tracked and visualized, followed by existing work on visual comparison aspects.

2.1 Data Science in Computational Notebooks

Computational notebooks support incremental and iterative analyses well, enabling analysts to edit, arrange, and execute small code blocks in "cells". Cells combine an input for code with the output it produces, and can be executed in any order. However, this execution of cells in any order compromises reproducibility.

Large-scale studies document that the order of cells in a notebook and their execution order do not match for more than a third of the notebooks analyzed [22, 49]. In addition, the execution order cannot be tracked by the notebook alone, as only the last state of a cell is visible. In most cases, public notebooks have multiple large gaps in the execution sequence appearing in the notebook, where it is unclear what analysts executed and if it still exists in the notebook [49]. Notebook environments with reactive code execution models avoid issues from out-of-order execution and gaps in the execution sequence as they recompute results when the user changes a cell. Thus, at least the most recent state of the notebook can be reproduced. However, the previous code and outputs are still replaced when changes are made, leading to code duplication and messy notebooks when comparing different approaches. Also, when encountering dead ends, analysts often fail to document intermediate steps [9, 54]. Thus, notebooks are generally unsuitable for documenting the provenance of the analysis process.

Exploring the history of a notebook was rated the most challenging task after deployment in the survey by Chattopadhyay et al. [9]. When asked about remedies for tracking the evolution of a notebook, analysts favored automated versioning of their code and outputs [9]. However, tracking provenance alone is not sufficient. The tracked information must also be easily retrievable and easily digestible, for example, by relating it to another version of the notebook. We conclude that analysts can benefit from techniques that allow them to easily compare visual and textual changes between notebook versions, such as different data versions or alternatives tested [3, 9, 54]. These requirements have guided the design of our approach.

2.2 Tracking and Visualizing Notebook Provenance

In the 2015 UX survey by the Jupyter organization [60], version control was one of the most desired features currently lacking in the Jupyter platform. Traditional version control management with tools like git is commonly used for notebook platforms such as Jupyter and Quarto [61]. However, these document-based notebooks represent visual outputs as encoded strings that frequently change, even if the content does not change, which makes the version history messy and confusing to navigate [9]. Further, traditional version controls cannot track the history of the outputs and metadata, like the execution order, which are not stored as part of the notebook structure. The Git extension for JupyterLab [30], based on nbdime [29], juxtaposes notebooks for comparison and shows differences in code and plain text. Commercial platforms, such as Observable or Google Colab, automatically track changes and store snapshots of notebooks on every change. JupyterLab creates a snapshot every time an analyst saves a notebook. Users can browse these notebook snapshots by date, but they do not provide details about the changes made in each snapshot. In Google Colab, users can also compare the textual content of two selected snapshots, such as code, markdown, or textual output. Comparisons of data or graphical output, including visualizations and images, are not supported. Observable notebooks allow forking to create variations of the analysis, yet lack features for direct comparison or reintegration of changes into the original notebook, which leads to messy workspaces with multiple copies of the same analysis [9].

ProvBook [55] is an extension to Jupyter Notebooks that tracks the provenance of cells. It enables users to compare individual cells to their previous versions by juxtaposing the previous code, the output it produces, and metadata, but without any explicit encoding of the differences. The effect of a change in one cell on other cells is not visible with this approach. MLProvLab [33] tracks the notebook's provenance and displays each version as a graph where the nodes are the notebook's cells and variables are the connecting edges. The graph's evolution over time can be inspected by browsing the executions. From the graph, a cell's code can be compared to earlier executions. MARG [52] visualizes a notebook as a graph with cells as nodes and connects them in the order of the analysis. This way, divergence points in the analyses, after which analysts try out multiple alternatives, can be displayed. The resulting graph is similar to provenance graphs of analysis processes in visual analytics tools [21, 24, 57]. The approach, however, requires manually annotated notebooks and provides no comparisons. AARDVARK [16] is a visual debugging method to identify and compare differences in the program flow and variable values after code modifications. Persist [17] tracks provenance for interactions in interactive outputs in notebooks, yet does not tackle overall notebook provenance.

Most closely related to our work is Verdant [31], which automatically records the provenance of analyses conducted in notebooks and visualizes it in two tabs: the activity and artifacts tab. Like the histories in Google Colab and Observable, the activity tab lists time-stamped revisions. For each revision, a barcode visualization summarizes the changes to the notebook. Selecting a revision opens the notebook's old state in a separate tab in JupyterLab. Verdant's artifact tab lists the cells and displays the input and output versions. Analysts can inspect the history of cells across all notebook revisions. In all views, only differences in the cell's code input are highlighted.

In contrast to the works described above, our visual representation of the notebook and its history resembles that of the source notebook. Instead of representing the notebook with a graph or barcode, we represent the notebook cells as rectangles aligned from top to bottom next to the notebook. In addition, we align the history with the cell that is currently active in the notebook. Loops' visualization of the notebook history shows the notebook structure, how it changes over time, if and how the cells have changed, who changed them, and summarizes the difference between the various types of content in the notebook, thereby providing a comprehensive view of the changes made.

2.3 Visual Comparisons

Comparison of different states of inputs and outputs is an essential task for understanding the evolution of notebooks. However, the types of content present in notebooks are diverse, and hence, efficient comparison methods have to be tailored to the type of content. This section discusses ways to compare notebook content visually, focusing on plain and rich text, code, data tables, and images. We do exclude dynamic content such as audio, video and embedded websites.

Gleicher et al. [19] describe the design space of comparative visualization as being made up of *juxtaposition*, *superimposition*, and *explicit representation*. *Juxtaposition*—presenting objects side-by-side or over time through animation—is often easy to implement and can be used for any visual representation. The downside of the approach is that it relies on memory for comparison. *Superimposition*—placing objects in the same coordinate system—is common for spatial data or comparisons of similar objects but can easily lead to clutter. *Explicit representation* directly encodes differences, eliminating the need for mental comparisons. To contextualize differences, explicit representation is frequently combined with superimposition or juxtaposition. In the following, we discuss visual difference encodings for the various content types present in notebooks.

Text difference visualizations are widely employed in text editors. Overleaf and Google Docs, for example, track and highlight textual changes for plain and rich text. Myers algorithm [41] is most commonly used to create human-readable text differences. The algorithm computes a "diff" by finding the shortest path in the edit graph for two strings. It is the default option of git and several code editors to show **code differences** [44]. While changes to plain and rich text are commonly visualized within the document with explicit encodings, most code editors default to juxtaposing the compared versions and explicitly encoding additions and removals.We support both in-document and juxtaposed comparison in Loops.

In addition to the layout considerations above, Gleicher [18] also points out that **image differences** could be identified at different abstraction levels: on the data, feature, and image level. The *data* level refers to the raw data to create the image, the *feature* level refers to the abstracted data (e.g., the height of a bar in a histogram), and the *image* level to the resulting imagery. As our approach operates on the visible output of Jupyter notebooks, we focus on related work that compares images (and data visualizations) at the image level.

Notebook environments and the related work discussed so far juxtapose graphical outputs in the notebook, but do not use explicit visual encoding of differences [18]. For superimposition, blending and color weaving have been proposed to combine information from layered images [19, 38]. To explicitly encode differences between images, a common method is to convert them to grayscale and create a difference image by subtracting the individual pixel values of one image from the corresponding pixel values of the other [20, p. 87–90]. Pixel differences can be color-coded, for example, in red and green for negative and positive differences, respectively [26]. However, global translations, antialiasing, or compression in lossy formats can cause large but irrelevant differences [26, 27, 53]. To suppress these artifacts, fuzzy difference methods can be applied that either take neighboring pixels into account [26], or allow a certain distance between the compared pixels in color space [27]. Resemble.js [53] employs a nuanced approach, calculating pixel-based image differences with options to ignore subtle changes, color differences, transparency, and antialiasing; it also provides several modes to visualize differences. VAICO [56] supports comparing more than two images simultaneously, using hierarchical clustering of regions of differences and superimposition of the results.

To identify differences in **data visualizations**, previous studies evaluated the efficacy of superimposition and juxtaposition for various tasks in crowd-sourced experiments [28, 45]. While participants performed best with animated and superimposed visualizations when estimating correlations or identifying the largest difference between visualizations, juxtaposed visualizations performed better for certain tasks [28]. Vis-a-Vis [5] visualizes the evolution and differences in visualizations in the context of changes in code and code structure. Vis-a-Vis displays the current source code, output, and a revision tree, grouping revisions by code structure. The interface displays outputs linked to a structure, along with a comparison image showing the outputs' per-pixel variance. In Loops, we juxtapose or superimpose two images (including images of data visualizations) and highlight pixel differences. We color-code pixel differences to indicate added, removed, or changed content. We also group neighboring changes by slightly dilating the pixel-based difference. If the images/visualizations are superimposed, we provide sliders to vary the opacity of the images dynamically.

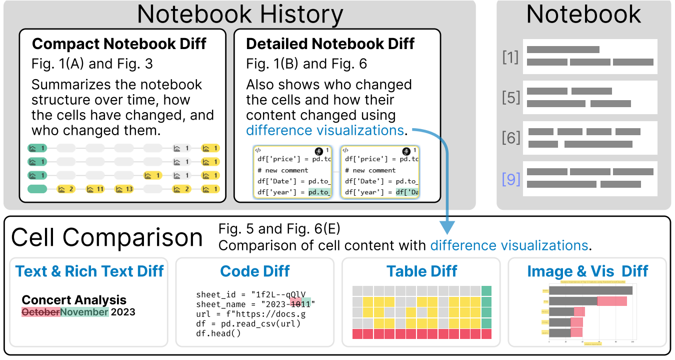
As data is fundamental to data science, **data differences** are at least as important as code differences. Datasets evolve over time as analysts filter, clean, update, or expand them during their work. However, data difference visualizations are rarely integrated into the tools used by analysts. While notebooks can be used to analyze arbitrary data set types, we focus on arguably the most common data type: tabular data. TACO [43] introduces an approach to visualizing changes in data tables. The technique categorizes changes as additions and removals, merges and splits, reorders, and content changes. TACO juxtaposes the compared tables and shows a heatmap of the differences, highlighting content modifications, additions, and removals through distinct colors. Furthermore, the changes are summarized in a histogram, offering an overview of each version. CHAMELEON [25] is a visual analytics approach that focuses on the impact that changes in data have on machine learning projects. They show a data version timeline, from which two versions can be compared in detail. Changes in the data's distribution are visualized with superimposed diverging histograms for each feature. In addition, a prediction change matrix and a sensitivity histogram explicitly show how the machine learning model's predictions are affected by the change in data. In prior work [13], we visualize data sets that change over time as time curves [4]. Juxtaposed summary visualizations and explicit difference visualizations show how the data changed between two versions, with differences sorted based on the extent of change. Perhaps most closely related to our work is Diff in the Loop [62], which contains a code editor that tracks changes to code and runtime variables. A separate view shows how the data set has changed through code edits by visualizing the distribution of all affected features.

In Loops, we explicitly visualize the differences in tabular data by color coding changes, additions, and removals. The resulting visualization is similar to the heatmap in TACO [43]. However, we do not incorporate color coding to indicate the magnitude of change within a table's cell, nor do we consider the reordering of columns a change. The effects that differences of the tabular that have on the subsequent analysis are not visualized directly but through the difference visualizations in the following cells as soon as they are re-executed.

3 Loops Approach

Data analysis is an iterative process. Based on the insights analysts gain during their analysis, they need to change their previous code and assess how the results are affected. With Loops, we not only track the notebook provenance for reproducibility of the analysis results, but also to support the ongoing analysis process. We aim to achieve this by visualizing how the notebook has changed in the series of loops that make up the analysis process. We show the changes made by analysts and their impact on the notebook outputs with difference visualizations (*diffs*) for the whole notebook and individual cells.

Loops design is guided by the issues that analysts face when working in notebooks (see Sec. 2.1). First, we help analysts document their analysis process, including all intermediate steps, by automatically recording the notebook's provenance. Second, exploring the history of a notebook was rated as one of the most challenging tasks [9]. We remedy this with Loops notebook history visualization, which reveals when and in what order cells were changed and executed, which cells were changed most frequently, and whether other collaborators have changed any cells. We summarize all this information for individual notebook versions in the compact notebook diff. A detailed notebook diff also reveals how cells were changed, addressing the third issue analysts face in notebooks: comparing alternatives. Currently, comparisons happen



Summarizes the notebook structure over time, how the cells have changed, and who changed them.

Also shows who changed the cells and how their content changed using difference visualizations.

```
df['price'] = pd.tc
# new comment
df['Date'] = pd.to_
df['year'] = pd.to_
```

```
df['price'] = pd.tc
# new comment
df['Date'] = pd.to_
df['year'] = df['Da
```

Comparison of cell content with difference visualizations.

Concert Analysis
~~October~~November 2023

```
sheet_id = "1f2L--qQlV
sheet_name = "2023-1011"
url = f"https://docs.g
df = pd.read_csv(url)
df.head()
```

Fig. 2: The notebook's history is presented next to the analyst's notebook, visualizing changes between notebook versions with compact or detailed notebook diffs. The cell comparison can be opened on demand to explore the differences in further detail.

by duplicating code, resulting in notebooks with outdated or redundant code [54]. We facilitate the comparison by placing the notebook history next to the notebook and aligning the diffs in the history with the currently active cell in the notebook (see Fig. 2). The detailed notebook diff shows difference visualizations for every change in the cells' input and output. This allows us to show the changes made by analysts, but also the impact of their changes on other cells. Loops also provides a full-width detail view for cell comparison, with which individual changes can be explored further (see Fig. 2).

In the following subsections, we explain how Loops tracks the notebook's provenance and how it is visualized and compared in its components. The supplementary video (see Sec. 8) demonstrates the workflow and interactions.

3.1 Tracking and Visualizing the Notebook History

Loops automatically records the interaction provenance, as requested by analysts in prior work [9]. Every time users execute a cell in the notebook and thus update it, Loops saves the notebook's state. Loops stores the cells of the notebook, their order, which cell was active and executed, who executed it, the cells' input (i.e., the code) and output, and how these inputs and outputs were displayed in the notebook–rather than the content of variables used in the execution. Consequently, Loops stores everything an analyst sees while working in the notebook and thus influences their information-foraging and sensemaking loops. The goal of Loops is not to collect comprehensive provenance that guarantees reproducibility but to support and make the iterative analysis process comprehensible. We argue that through this support, many issues that often lead to unreproducible notebooks can be mitigated (see also Sec. 1).

As with most provenance data, the scale of interaction histories can become uninterpretable and overwhelming quickly [9]. Therefore, we considered how the provenance can be best aggregated. If at all, previous work has grouped notebook versions by temporal proximity [32]. For Loops, however, we decided to take a different approach that leverages the semantics of notebooks: Loops groups the stored notebooks states as long as their cells are executed in linear order (see Fig. 3). This means that the notebook provenance is aggregated as long as the analyst executes the notebook from top to bottom, even if the analysis is interrupted between executions.

Although the analysis process is typically not linear, its steps still build on each other: Data must be loaded before exploration, cleaned before modeling, etc. These dependencies are also reflected in the structure of the notebook [52]. Going back in the notebook—and re-evaluating already executed parts—corresponds to a new iteration of the analysis loop. In Fig. 1 Ⓐ, for example, a new version is introduced as the analyst edits and re-executes the first cell of the notebook again before updating the data processing step and re-running the modeling step with the new data. The aggregated notebook provenance thus reflects the analysis loops we visualize and compare in Loops.

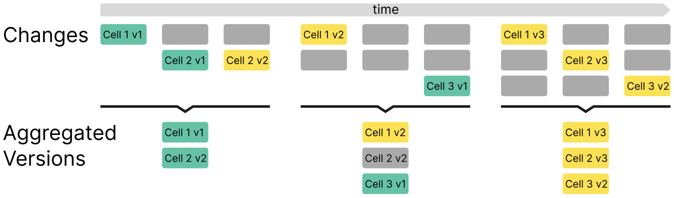


Fig. 3: Aggregation of notebook changes to the versions visualized with Loops. Each column of changes corresponds to an execution of a cell. To simplify the provenance visualization, changes are aggregated into the same version as long as the notebook is executed from top to bottom. Note that no information about the structure of changes is lost.

This methodology to record and aggregate the provenance of notebooks is designed primarily to address the challenges encountered in sequentially executed notebooks (see Sec. 2.1). However, we argue that our approach is also valuable for analyses within reactive notebooks (see also Sec. 7).

The versions are visualized and arranged horizontally according to their creation order. Each cell is represented as a rectangle with rounded corners, reminiscent of a notebook cell. Like in a notebook, the cells are arranged vertically. Cells of adjacent versions are connected by thin gray lines, which serve as a visual guide for tracing cells across the notebook's versions. This is especially helpful if cells are not aligned (see Fig. 4). In addition, we align the cells of each visualized version around the cell that is currently active in the notebook. When the analyst changes the active cell, our provenance visualization realigns the cell's history vertically to enable better comparison (see Fig. 6 Ⓔ).

Notebook versions can be visualized at two levels of detail: (1) The compact notebook diff provides an overview of the version, indicating added, changed, executed, and deleted cells. (2) The detailed notebook diff also reveals the cells' content and how they have changed compared to the previous version. Below the notebook diffs, Loops displays a bar chart with a continuous time scale showing the temporal distribution of the notebook versions and the extent of their changes, indicating active and dormant periods of analysis (see Fig. 4). By visualizing the structure of the notebook, how and when it changed, and who changed it, we address the analysts' need for a retrievable and comparable history [9].

3.1.1 Compact Notebook Diff

The compact notebook diff, shown in Fig. 4, gives an overview of the evolution of the notebook, showing all cells in the same small size. We embed icons that indicate whether a cell contains markdown or code. If a code cell produces outputs, we encode the output type, i.e., whether it is a table or a visualization/image. Furthermore, an execution counter displays how often the cell was executed in each version. This helps identify the parts of the notebook that have been modified the most or not executed at all. Cells are color-coded according to their status: unchanged, changed, added, or deleted relative to the previous version. In accordance with common comparison tools for text and code, we use color-blind safe shades of ● green and ● red for additions and removals, ● yellow for content changes, and ● gray for unchanged content. Furthermore, the currently active cell in the notebook is emphasized with a ● blue shadow (see Fig. 4). When multiple analysts contributed to a notebook, we indicate each version's contributors at the top of the difference visualization (see Fig. 4). Contributors are sorted according to the number of executions they performed.

The compact cell design allows us to display 30 cells on Full HD screens without scrolling. This allows us to present the majority of notebooks in their entirety, as prior research indicates that more than 50% of all notebooks comprise 30 cells or fewer, and 95% contain 100 cells or fewer [54]. Fig. 4 shows an analysis carried out over several weeks with 211 executions visualized in 27 versions.

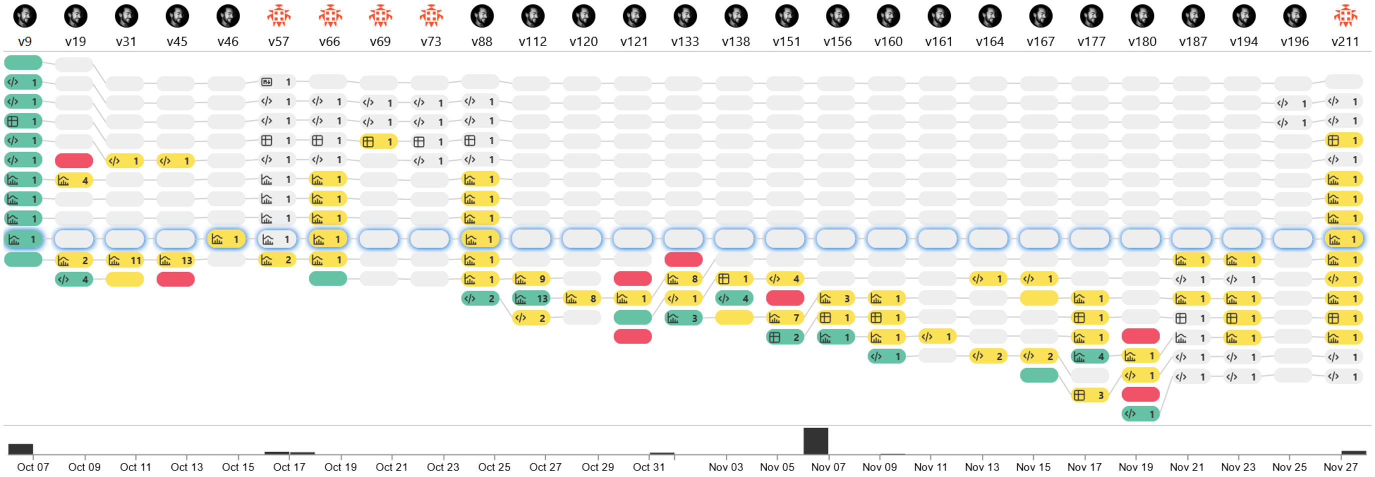


Fig. 4: Notebook history with compact notebook diffs for an analysis conducted over several weeks. Rectangles represent the notebook cells, which are color-coded to show insertions (● green), removals (● red), change (● yellow), and no changes (● gray). For executed cells, we show how often each cell was executed plus an icon indicating whether the cell contains M↓ markdown or </> code. For code cells, we change the icon if the cell outputs a ⊞ table or 📈 visualization/image. The cells of the versions are aligned around the active cell, emphasized with a blue shadow In the initial version on the left, the first analyst (👤) created and executed multiple cells. Later, mainly the cells at the end of the notebook were edited iteratively, and the entire notebook was executed again only a few times, primarily when the analyst changed in versions 57 (👾), 88 (👤), and 211 (👾).

3.1.2 Detailed Notebook Diff

The detailed notebook diff, shown in Fig. 1, visualizes the changes in both code and output of a cell, allowing analysts to understand their changes' impact better. We adapt the displayed cell content depending on its output and change to give a better overview of a notebook version.

The content of markdown cells with headlines and code cells that output visualizations or images are always shown, as these elements provide structure to the document, and showing them facilitates navigation [9,54]. However, we limit the cell content shown to the markdown cells' headings and the code cells' visualizations/images, as long as other cell parts are unchanged. If there are no changes, code or outputs that don't generate a visualization have their content hidden. Cells where neither the code nor the output are displayed are represented with a small rectangle, as in the compact notebook diff. This approach allows the notebook diff to focus on changes and their impact across the notebook (see Fig. 1 Ⓑ). If there are changes in code or outputs, a superimposed difference visualization for the respective cell content is displayed (described in Sec. 3.2). The code and the outputs of the currently active cell are always displayed so that the history of the cell can be easily tracked and compared, as illustrated in Fig. 6 Ⓔ. Because we display the content of cells in the detailed notebook diff, we shift from color-coding the entire cell to color-coding only the cell border. This adjustment addresses contrast issues and ensures the cell content remains clearly visible.

In addition to its content, we show the cell's type and how often it has been executed in the top left and right corner, respectively, mirroring the compact notebook diff. Furthermore, we show who contributed to the changes in the cell. Contributors are represented by avatars adjacent to the execution count and are arranged based on the number of executions they have performed (see Fig. 1).

3.2 Visualizing Cell Content Differences

For a comprehensive comparison of the cell content, analysts can open a separate detail view for cell comparison from each cell in the history. We show this view below the notebook to leverage horizontal space that is useful for side-by-side comparisons (see Fig. 6 Ⓒ).

The cell comparison can be used for either code or output and features large difference visualizations that display the content of the cell and how it changed (see Fig. 5). The view automatically shows an appropriate difference visualization based on the content type. We have carefully designed visual encodings that consistently visualize changes in the cell's code and the various types of content present in the cell's output. All difference visualizations can display either juxtaposed or superimposed comparison, offering flexibility in viewing and comparing changes. In both layouts, changes are explicitly encoded by applying the same color scale used for cells.

Next, we introduce the difference visualizations tailored for various content types. These visualizations are employed not only in cell comparison but also in the detailed notebook diff, albeit in a more condensed format.

3.2.1 Text and Code Comparison

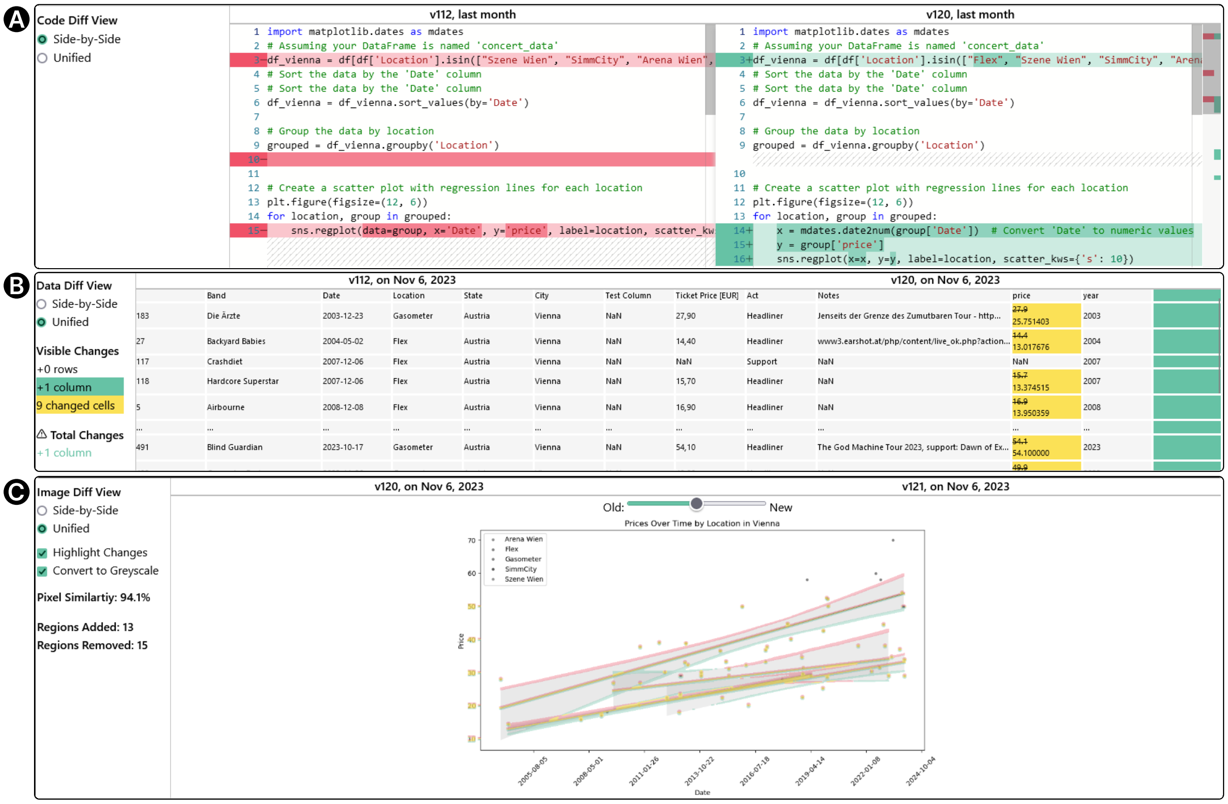
For the visualization of text and code differences, we based our designs on standard methods used in word processors and editors. Our difference visualization employs the Myers algorithm [41], which is widely used to detect insertions and deletions in text. Myers difference algorithm is particularly effective for comparing text documents, as it can efficiently identify the longest common subsequence of words or lines between two texts, making the display intuitive to understand. We use this approach for plain and rich text.

For visualization of code differences, we use a dedicated code editor. This editor provides syntax highlighting and employs the Myers difference algorithm but with additional post-processing steps applied to the detected changes (see Fig. 5 Ⓐ). These additional steps account for the unique characteristics that code presents in contrast to text documents to ensure that the differences are meaningful and relevant. Alternative algorithms have been shown to provide better results for code than the standard Myers algorithm, but prior work has not considered post-processing steps [44]. Given its frequent use in coding tools and developers' ensuing familiarity, we chose this combination over alternatives.

3.2.2 Tabular Data Comparison

Understanding how tabular data changes over the course of an analysis is crucial. Changes may be introduced through code or updates of the data source [3,25,40].

In Loops, these changes are explicitly visualized. Inspired by TACO [43], our visualization compares two tables as they are output in the notebook. It identifies changes within the cells of a table and also detects the addition or removal of rows and columns. Columns or rows that are added or removed are highlighted in green and red, respectively. Changing the order of columns is not considered a change, as the columns' data remain unchanged. Cells in a table with changed content are highlighted in yellow; removed or newly added content is juxta-



Code Diff View

● Side-by-Side

○ Unified

v112, last month

```
import matplotlib.dates as mdates
# Assuming your DataFrame is named 'concert_data'
-df_vienna = df[df['Location'].isin(["Szene Wien", "SimmCity", "Arena Wien",
# Sort the data by the 'Date' column
# Sort the data by the 'Date' column
df_vienna = df_vienna.sort_values(by='Date')

# Group the data by location
grouped = df_vienna.groupby('Location')
-

# Create a scatter plot with regression lines for each location
plt.figure(figsize=(12, 6))
for location, group in grouped:
-    sns.regplot(data=group, x='Date', y='price', label=location, scatter_kw
```

v120, last month

```
import matplotlib.dates as mdates
# Assuming your DataFrame is named 'concert_data'
+df_vienna = df[df['Location'].isin(["Flex", "Szene Wien", "SimmCity", "Arena
# Sort the data by the 'Date' column
# Sort the data by the 'Date' column
df_vienna = df_vienna.sort_values(by='Date')

# Group the data by location
grouped = df_vienna.groupby('Location')

# Create a scatter plot with regression lines for each location
plt.figure(figsize=(12, 6))
for location, group in grouped:
+    x = mdates.date2num(group['Date'])  # Convert 'Date' to numeric values
+    y = group['price']
+    sns.regplot(x=x, y=y, label=location, scatter_kws={'s': 10})
```

Data Diff View

○ Side-by-Side

● Unified

Visible Changes

+0 rows

+1 column

9 changed cells

⚠ Total Changes

+1 column

v112, on Nov 6, 2023 | v120, on Nov 6, 2023

	Band	Date	Location	State	City	Test Column	Ticket Price [EUR]	Act	Notes	price	year	
183	Die Ärzte	2003-12-23	Gasometer	Austria	Vienna	NaN	27,90	Headliner	Jenseits der Grenze des Zumutbaren Tour - http...	~~27.9~~ 25.751403	2003	
27	Backyard Babies	2004-05-02	Flex	Austria	Vienna	NaN	14,40	Headliner	www3.earshot.at/php/content/live_ok.php?action...	~~14.4~~ 13.017676	2004	
117	Crashdiet	2007-12-06	Flex	Austria	Vienna	NaN	NaN	Support	NaN	NaN	2007	
118	Hardcore Superstar	2007-12-06	Flex	Austria	Vienna	NaN	15,70	Headliner	NaN	~~15.7~~ 13.374515	2007	
5	Airbourne	2008-12-08	Flex	Austria	Vienna	NaN	16,90	Headliner	NaN	~~16.9~~ 13.950359	2008	
...	...	...	...	...	...	...	...	...	...	...	...	
491	Blind Guardian	2023-10-17	Gasometer	Austria	Vienna	NaN	54,10	Headliner	The God Machine Tour 2023, support: Dawn of Ex...	~~54.1~~ 54.100000	2023	
										~~49.9~~		

Image Diff View

○ Side-by-Side

● Unified

☑ Highlight Changes

☑ Convert to Greyscale

Pixel Similartiy: 94.1%

Regions Added: 13

Regions Removed: 15

v120, on Nov 6, 2023 | v121, on Nov 6, 2023

Fig. 5: Detailed code, table, and image/data visualization differences. The code difference **A** shows that the plotted subset was extended and that the data for the visualization's *x* and *y* axes were preprocesed further. The table difference visualization **B** shows that the concert data's price column was adjusted and a empty column was added. The data visualization difference **C** shows changes in the y-axis labels and regression lines.

posed within the cell, with the removed text being struck through (see Fig. 5 **B**).

3.2.3 Comparing Images and Visualizations

Images and data visualizations can be compared at multiple levels: on data, feature, and image level [18]. In Loops, we compare them at the image level (and assume visualizations are rendered as raster graphics) and thus do not differentiate between images and data visualizations in the following. While the comparison is made at the image level, the image difference visualization is part of a broader context that includes differences of all other steps preceding the image change. The difference visualization we propose aims to highlight smaller changes in particular, as larger changes—such as changes to the underlying data, the plot type, or one of its encodings—are also reflected in the data or code differences. Appendix A.3 contains a gallery of difference visualizations for several types of visualizations and changes in them.

Balancing the needs for comparison between generic images (photos, diagrams) and data visualizations is difficult. For instance, a slight change in brightness may not matter in a photograph but can indicate a crucial change to the data underlying a heatmap. Given the data science focus of this work, we prioritize the requirements for comparing data visualization over those for generic images.

Image subtraction is used routinely for enhancing differences between images [20, p. 87] and is useful when changes in successive images need to be detected [8, p. 429]. This is the primary use case of our work: to highlight changes between iterations. We use pixel-based subtraction to create the difference visualizations for images. The process is illustrated in Appendix A.1.

If the image sizes vary, they are first scaled, preserving their aspect ratio, and subsequently padded to the same size. We pad to the right and top because data visualizations usually align axes with the bottom left. To determine the color for the added pixels as part of the padding process, we extract the background color by analyzing the outermost 1% of pixels on all four borders of the image, and assume that the most frequent color is the background color. The images are then subtracted and subsequently turned into grayscale by calculating the perceived luminance of each pixel to preserve the color differences [8, p. 256]. Thresholding converts the grayscale image into a binary image, by setting all pixels exceeding a 1% change in luminance to white (changed) and all others to black (unchanged). As the pixel-based approach may result in many small disjoint changes, we apply additional morphological operations to make the difference visualization more comprehensible, akin to summarizing text changes on a word rather than a character level. Differences are dilated twice, which merges adjacent differences into one, using a 3 × 3 kernel for the detailed comparison and a 9 × 9 kernel for the detailed notebook diff to ensure visibility in the scaled-down representation. Afterwards, the differences are eroded once, using the same kernel, to reduce the size of the differences again. Performing the erosion only once (when we dilated twice) results in the difference being slightly larger than the changed content, forming a small border around the changed pixels. This approach is inspired by the background color used to mark changes in text, providing a clear and distinct visualization of modifications (compare Appendix A.3). The calculated difference is then transformed into an RGB image. Pixels that changed from the background color to a different color are encoded in green, from a non-background color to the background color in red, and pixels that changed color in yellow (see Fig. 6 **E**). We also calculate the changes' bounding boxes. In initial experiments, we highlighted changes using rectangular or convex hull bounding boxes, but within the bounding boxes the changes were challenging to identify. Instead, we now count the bounding boxes to inform the user about the number of changed regions in the image, first removing any bounding boxes nested within others, to highlight subtle changes as in the scatterplot in Fig. 12.

Users can choose whether the images with explicit difference encoding are displayed juxtaposed or superimposed. In the superimposed difference visualization, the images are combined with alpha blending, and the opacity levels can be controlled with a slider (see Fig. 5 **C**).

Whether juxtaposition or superimposition is better is task-dependent [19, 28, 45]. Generally, superimposition is most helpful if the compared images are similar, while significant changes are better represented in juxtaposed views [19,28,45]. We therefore superimpose highly similar images and juxtapose dissimilar images. To determine similarity and decide on the layout to use by default, we evaluated multiple image similarity metrics: (*i*) Mean Squared Error (MSE), (*ii*) Structured Similarity Index (SSIM) [65], (*iii*) Normalized Mutual Information (NMI) [58], and (*iv*) Oriented FAST and Rotated BRIEF (ORB) [7]. We excluded MSE and NMI from further consideration due to a lack of interpretability of the resulting value. The results of SSIM and ORB are promising, but to avoid a discrepancy between the visualized differences and the calculated similarity we opted for a more straightforwrd approach based on our explicit encoding of differences. We divide the number of changed pixels by the total number of pixels in the image to obtain a relative pixel similarity measure. Further details on the similarity metrics and the results of our experiments are provided in Appendix A.2.

In addition to guiding the default representation (superimposed vs juxtaposed), we also show the numerical value in the difference visualization. We present the difference visualization superimposed as long as the similarity is at least 90%. Otherwise, we use the juxtaposed layout. If the similarity of pixels is less than 75%, we remove the explicit encoding of differences, assuming that the images are different enough to be recognized without visual aids. Our experiences match findings by Lyi et al. [37] who found that "explicit-encoding can be used for designs where visualizing subtle difference is of importance".

4 Implementation

We have implemented the open source prototype of Loops as an extension for JupyterLab that is available on GitHub: `https://github.com/jku-vds-lab/loops/`. A deployed instance of JupyterLab with Loops is available at: `https://mybinder.org/v2/gh/jku-vds-lab/loops/main`; this instance also contains example notebooks. Although the screenshots and use cases presented in this work use Python, the Loops prototype can be used with any programming language available in Jupyter, as it only operates on Jupyter's frontend and has no dependencies on the notebook's kernel.

We use Trrack [11] to store the notebook states as a provenance graph. For the difference visualizations, we use html-diff [59] for rich text differences, the Monaco Editor [39] for code differences, D3 [6] to create the table difference visualizations, and OpenCV [7] to create the image differences.

5 Use Cases

We demonstrate Loops by means of two use cases. The first use case presents an analysis of Austrian concert data conducted over several weeks and demonstrates Loops' visualization of the analysis history. The second use case demonstrates how we support analysts in comparing the results of a what-if analysis on data from lung cancer patients. The notebooks of the two use cases, together with the provenance of the analysis, are available in the deployed instance of Loops: `https://mybinder.org/v2/gh/jku-vds-lab/loops/main`. Furthermore, we collected qualitative feedback from notebook users with various backgrounds, which we summarized in Sec. 6.

5.1 Use Case 1: Multi-Week Concert Data Analysis

Our use case presents an analysis of Austrian concert data conducted over several weeks and demonstrates Loops visualization of a long analysis process by two analysts (see Fig. 4). The data set is a long-form table with 501 entries from 2003 to 2023. Its features describe the artist, date, location, ticket price, and act (headliner/support). We load the concert data from an external Google Sheets spreadsheet, which is updated regularly and allows us to show differences that are not caused by code changes. The history contains 211 executions represented in 27 versions. The final notebook consists of 16 cells, a typical size for notebooks [54].

Analyst ● starts the analysis on October 6th by importing the concert data and applying a few initial data wrangling steps. They then profiled the tabular data to examine available years and how distributions change over time. The primary goal of the analysis was to investigate the increase in ticket prices over the years. While they created various plots using different visualization techniques, it was necessary to repeatedly return to the data wrangling step to update the data types to fit the plot (v19–v46 in Fig. 4). After updating the data, the visualization cell was changed up to 13 times in one version.

The analyst ✲ re-executed the notebook on October 16th (v57) and 17th(v66). The detailed notebook diff reveals no code changes, but outputs were changed due to external data updates. As cell outputs are overwritten upon execution, investigating how an output changes with new data becomes challenging, particularly with multiple outputs in the notebook. In this context, Loops enhances our ability to recognize and track changes by clearly highlighting output differences.

The most extensive phase of the analysis occurred on November 6th, involving 118 of 211 versions, visible in the bar chart in Fig. 4. Analyst ● extended the visualization of price trends over time with regression lines and adjusted prices for inflation by including a data set containing inflation rates. Adjusting prices for inflation required several iterations, and we used Loops to verify the changes using the difference visualizations shown in Fig. 5. In the last version of the notebook from November 27th (v211), analyst ✲ executed the entire notebook, which caused the visualizations to change due to the updated concert and inflation data.

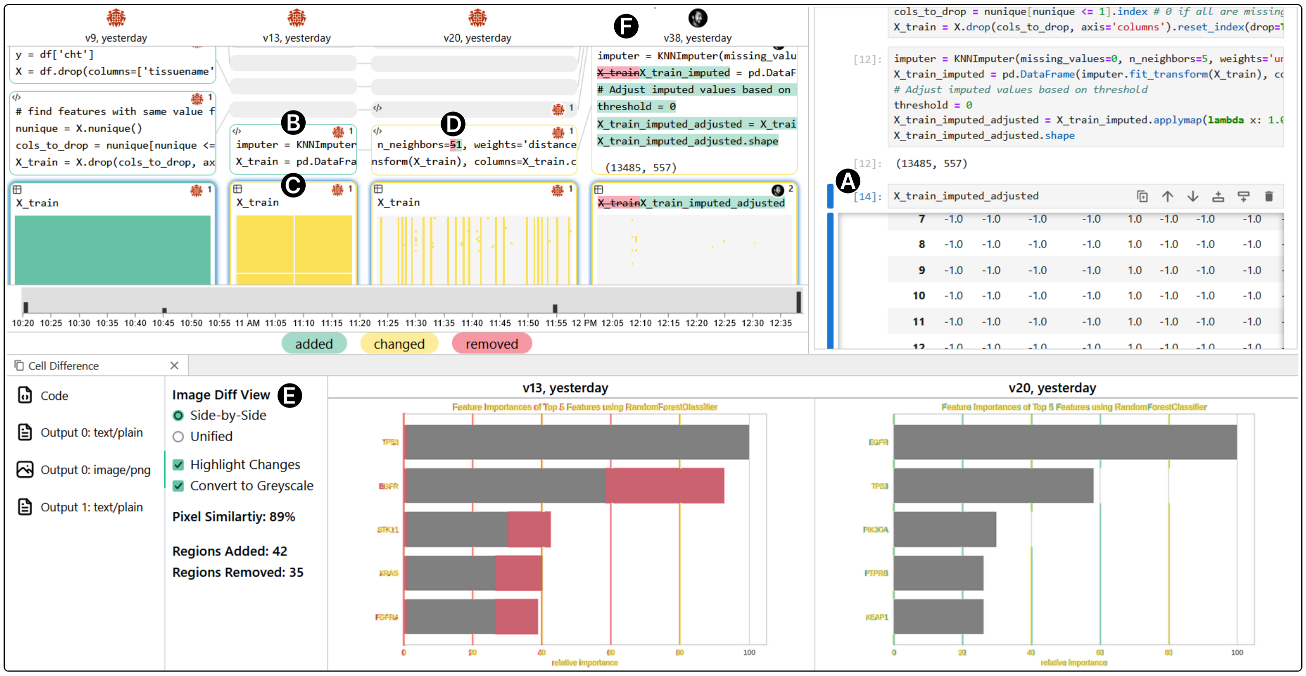
The history of the notebook shown in Fig. 4 provides several insights: (1) This first analysis steps by analyst ● are represented in the first five versions of Fig. 4 that summarize 46 notebook versions. They show that after creating and executing the cells with basic visualizations of distributions, most changes and executions aimed at visualizing the price trend. This involved repeated cycles of changing the data format and visualizations. (2) Versions in which we resumed the analysis after a break are evident as the notebook was re-executed from the top, ensuring that required libraries and the data are loaded. This can be seen in versions 57, 66, 88, and 211 of Fig. 4 that correspond to October 16th, 17th, November 6th, and 27th, respectively. (3) Versions 57, 88, and 211 additionally indicate a transition to a different analyst. (4) The history also shows that during the extensive analysis on November 6th (v88–v194), no changes were made to previously created cells.

5.2 Use Case 2: What-If Analysis on Cancer Patient Data

The analysis in this use case builds on our previous work comparing lung cancer patient cohorts from the AACR Project GENIE [1, 12]. We aimed to validate the mutational differences between these cohorts, as reported in the literature [42]. However, we found that the cohort data also considerably varied in the amount of missing data. Our approach identified mutations in the FGFR4 gene as a key feature to differentiate between patient cohorts, although they differed primarily by the amount of missing data. We hypothesized that this gene would have minimal impact if the data were complete. In this analysis, we now want to go beyond the available data and perform a what-if analysis, substituting the missing data [36].

The first analyst ❖ begins the analysis by replicating our previous findings. They import the necessary libraries, load the cancer patient cohort data, and inspect the data set. As the goal is to identify differences between cohorts using a random forest model, they separate the features used for the prediction and the cohort labels into distinct variables. They perform basic feature selection, eliminating features with the same value for all patients, as these do not provide any information for our model. After this, they again inspect the data set to verify the changes. They also create a visualization to show the distribution of FGFR4 genetic mutations among the analyzed cohorts. After preparing the data, they set hyperparameters, train the random forest model, and output its accuracy. They create a bar chart to visualize the model's most important features, and the results align with our previous analysis.

Next, the analyst substitutes the missing data and assesses how the results of the analysis change. They extend the initial feature engineering step and substitute the missing genetic data with a nearest-neighbor approach using the data of the five patients with the most



```
y = df['cht']
X = df.drop(columns=['tissuename'
```

```
# find features with same value f
nunique = X.nunique()
cols_to_drop = nunique[nunique <=
X_train = X.drop(cols_to_drop, ax
```

```
imputer = KNNImputer
X_train = pd.DataFra
```

```
n_neighbors=51, weights='distance
nsform(X_train), columns=X_train.c
```

```
imputer = KNNImputer(missing_valu
~~X_train~~X_train_imputed = pd.DataF
# Adjust imputed values based on
threshold = 0
X_train_imputed_adjusted = X_trai
X_train_imputed_adjusted.shape
```

```
(13485, 557)
```

```
X_train
```

```
~~X_train~~X_train_imputed_adjusted
```

```
cols_to_drop = nunique[nunique <= 1].index # 0 if all are missing
X_train = X.drop(cols_to_drop, axis='columns').reset_index(drop=T
```

```
[12]: imputer = KNNImputer(missing_values=0, n_neighbors=5, weights='ur
      X_train_imputed = pd.DataFrame(imputer.fit_transform(X_train), co
      # Adjust imputed values based on threshold
      threshold = 0
      X_train_imputed_adjusted = X_train_imputed.applymap(lambda x: 1.0
      X_train_imputed_adjusted.shape
[12]: (13485, 557)
[14]: X_train_imputed_adjusted
```

```
 7   -1.0   -1.0   -1.0   -1.0   1.0   -1.0   -1.0   -1.0
 8   -1.0   -1.0   -1.0   -1.0   1.0   -1.0   -1.0   -1.0
 9   -1.0   -1.0   -1.0   -1.0   1.0   -1.0   -1.0   -1.0
10   -1.0   -1.0   -1.0   -1.0   1.0   -1.0   -1.0   -1.0
11   -1.0   -1.0   -1.0   -1.0   1.0   -1.0   -1.0   -1.0
12   -1.0   -1.0   -1.0   -1.0   1.0   -1.0   -1.0   -1.0
```

Image Diff View

● Side-by-Side
○ Unified

☑ Highlight Changes
☑ Convert to Greyscale

Pixel Similartiy: 89%

Regions Added: 42
Regions Removed: 35

Fig. 6: Analysis of cancer patient data in JupyterLab with Loops. The cells in the notebook history are aligned with currently active cell in the notebook Ⓐ. The visualization of the notebook's evolution shows four versions corresponding to the initial execution of the notebook and the following what-if analysis. Analyst adds a cell to substitute missing values Ⓑ, which changes the data Ⓒ. They subsequently update the code in version three Ⓓ and retrain the model with the new data, which changes the features that are most important to distinguishing cohorts Ⓔ. Analyst then updates the code for substituting missing values again in the final version of the notebook Ⓕ.

similar genetic profile (Fig. 6 Ⓑ). As they output the new data, all of the cells are marked as changed (Fig. 6 Ⓒ), due to a change in data type by averaging the nearest neighbors data. They therefore change the code to only use the data of the most similar patient (Fig. 6 Ⓓ). After re-executing the subsequent cells, the detailed notebook diff shows that the bar chart changed significantly for the new model. The analyst opens the cell comparison (Fig. 6 Ⓔ) and sees that FGFR4 is no longer one of the most important features distinguishing cohorts. Also the overall order of important features has changed significantly, as highlighted by the bars' altered labels. EGFR now ranks as the most significant gene, followed by TP53 and PIK3CA. KEAP1, and PTPBR have been added to the list, while STK11 and KRAS have been removed (Fig. 6 Ⓔ).

In the last version of the notebook (Fig. 6 Ⓕ), another analyst reworked the data substitution. They update the code so that the data of multiple patients are considered, as analyst had planned, while keeping the same data type. However, this change results only in minor changes of the data and the model (see Fig. 17).

This use case exemplifies our approach to test and compare different analysis paths. The same procedure can analyze the effects of different substitution methods, models, or hyperparameters. The AACR Project GENIE is an ongoing effort with continuous data updates, so the analysis will need to be repeated in the future. With Loops, any resulting output differences can be quickly identified.

6 Qualitative User Feedback

We gathered feedback on our approach by interviewing notebook users with various backgrounds: a student (S) in AI, a professional data scientist (DS), a researcher (R) working in human-centered AI, and a lecturer (L) teaching undergraduate and graduate students in AI. We recruited the participants from our professional network, targeting individuals proficient in computational notebooks. We emailed invitations to individuals and research groups until we could recruit a representative for each user group. All participants have a minimum of three years of experience working with notebooks. The interviews lasted between 40 and 60 minutes and were guided by a set of initial questions provided in Appendix C. We encouraged participants to provide think-aloud feedback, fostering a natural conversation flow. Despite this, we ensured that all questions were addressed during the interview. Each interview was conducted individually and recorded, after which the recordings were reviewed to take detailed notes on the interviewees' comments, feedback, and responses. The interviews with DS and R were conducted remotely, while the interviews with S and L were conducted in person. Participants were not compensated for their involvement.

In the interviews, we first asked them questions about how they work in notebooks (see Appendix C). Participants S, DS, and R stated that they commonly create multiple cells, each containing code with slight variations to facilitate the comparison of alternatives. When asked how they deal with alternatives once they had picked a solution, none of the participants could articulate the specific criteria they apply. S and DS leaned towards retaining all code, even if it resulted in cluttered notebooks. Notably, all four participants had experienced situations in which they would have needed code they had previously deleted. The four notebook users stated that they mainly work alone on notebooks; collaboration only takes place asynchronously, if at all. These responses are consistent with the results of previous studies [3, 9, 54].

As a next step in the interview, we showed them a notebook we had prepared based on the concert data analysis of Use Case 1. We explained the data, the analysis goals, and the notebook structure. We then opened the compact representation of the analysis history. L appreciated how the history lets them track activities in the notebook over time. Participants stated that the color-coding with green for new and red for removed cells was obvious, yet, they were unsure about the meaning of yellow cells, triggering us to add a legend to Loops. They liked that the cells' execution can be tracked, as they regularly experience issues with out-of-order execution and needed to find out how they got to a particular state. When asked which cells had changed the most, all participants could quickly identify the corresponding cells.

After exploring the compact notebook diff, we started executing the notebook, showing how the history aligns with the currently active cell and how it is updated as cells are executed. The execution of the notebook with data that have been updated since its last execution (see Use Case 1) resulted in changed outputs. Since participants were not yet familiar with the data, they could not tell how outputs changed from the notebook alone. We changed the last notebook version to the detailed notebook diff, revealing the changes in data and visualizations. S and DS appreciated how the extension summarizes the individual versions of the notebook. DS added that Loops simplifies analyzing the impact of changes. They also appreciated that the differences are directly visible without using git, and that difference visualizations are available for all content types. R appreciated that previous outputs remain visible, enabling them to revisit them during long-running executions or after unintended executions.

We also opened the comparison detail view for multiple cells to demonstrate the detailed text, code, table, and image difference visualizations. Participant S highlighted the comparison detail view as a great addition to the notebook representation. The participants recognized how the familiar encoding of code differences was translated to other types of content. They consider all difference visualizations to be most useful when changes are subtle. The flexibility to toggle between superimposed and juxtaposed comparisons was also appreciated. For the image difference visualization, DS liked how images are shown in grayscale to highlight the changes with color. S noted that it may surprise her when she sees that a chart has changed significantly, e.g., due to a change in chart type. However, she also added that she could recognize this change based on the differences in the preceding code difference. Participant R often creates large composite figures and hence found the image difference visualization to be particularly useful when styling these figures to pinpoint elements that still require updates.

Overall, participants were enthusiastic about the change summaries during analysis and appreciated that the notebook's provenance was recorded, ensuring no loss of code or results. They also found the visualization of notebooks and cell content differences intuitive. However, participants also noted some concerns and weaknesses. They expressed a desire to extend the examination of differences beyond the visible contents of the notebook, e.g., to changed variables (R) or to the full data set when it is truncated in the notebook (DS, R, L). The current version of Loops only indicates that the size of the table changed next to the difference visualization of the visible content. S expressed concerns about balancing the size of difference encodings in image difference visualizations for small and large changes. L would value the ability to analyze changes from the most recent execution rather than the aggregated changes from the last version of the notebook. R and S suggested an onboarding process to explain the aggregation, Loops's features and, most importantly, interactions. Furthermore, R emphasized the utility of showing a notebook history in collaborative scenarios, allowing users to track changes made after handing over the notebook. Similarly, S expressed a desire for transparency in collaborative work, suggesting a feature that identifies change contributors.

7 Discussion

We have demonstrated the utility of our approach in the use cases and received positive feedback from potential users. In the following, we first summarize the scalability limitations of our approach regarding the notebook's history and content, followed by a discussion on the limitations concerning collaboration and reactive notebooks.

Scalability. The compact notebook diff scales well with an increasing number of cells. However, the number of versions depends on the user's execution patterns, and horizontal space is limited. Visualizing code differences within the notebook diff also faces spatial constraints, often requiring users to scroll horizontally within the cell to view differences fully. Our attempts to reduce code differences to the neighborhood of changed content encountered challenges. Especially when exploring a cell's history, the individual changed code parts should remain comparable. To address this, we are considering a more nuanced degree-of-interest function to control the amount of visualized code in a changed version and its adjacent versions [2].

In the notebook diff, the pixel-based difference visualization for tabular data faces similar spatial constraints, as Niederer et al. [43] already discussed for TACO. Due to the limited horizontal space, only a few hundred columns can be displayed without scrolling. However, we argue that our visualization scales significantly better than the output in the notebook itself (see Fig. 18). The detailed difference visualization of tabular data shown when comparing a cell also displays the table content and faces the same constraints as the table output in the notebook. The pandas library, which we use in our cases to load and manage data, by default truncates tables to 10 rows and 20 columns when output in the notebook. Our difference visualization relies on this output, so only changes within this subset are shown. This has led our users to express a desire to see a full diff (see Sec. 6). However, this would require implementations tailored to different programming languages and environments.

Our difference visualization for images and data visualizations also has limitations. We opt for a detailed pixel-based approach because the difference visualization is part of a broader context in the notebook diff, from which substantial changes in a visualization can already be anticipated. But the pixel-based difference approach is sensitive to images of varying sizes or internal shifts. Fig. 6 encodes the title, grid lines, and x-axis as changed due to a slight shift to the left. To address these issues, data visualizations could greatly benefit from a dedicated difference analysis approach, akin to those developed for fields like historical art [48] or fluid dynamics [46]. Such an approach would need to be aware of the visualized data, its representation, and the comparison task to ensure that the differences are meaningful. For example, during our interviews, one user expressed interest in seeing differences in styling specifically (see Sec. 6). For other tasks, only changes that affect the marks may be relevant. Given the multitude of existing visualization types and design choices, exploring this in depth is beyond the scope of this paper.

Collaboration. Collaboration and coauthoring of notebooks are crucial aspects of data analysis [36]. The notebook users we interviewed rarely collaborate with colleagues on a notebook. And when they do, they take turns working with them on the notebook. Loops already effectively supports such asynchronous collaboration by visualizing contributors to notebook versions and cell changes. For real-time collaboration, the suitability of our approach to aggregate notebook provenance varies depending on the collaboration style. In a study by Wang et al. [63], half of the teams collaborating in real time split their tasks to explore alternate solutions. Editing and executing code in different parts of the notebook simultaneously can result in more notebook versions than the analysts intended.

Reactive Notebooks. Some scholars argue that reactivity is a prerequisite for real-time collaboration [47]. Reactive notebooks automatically update the output as the data used by these outputs change, removing concerns about out-of-order executions. We still see advantages in using Loops with reactive notebooks. First, modifying a single cell can immediately affect multiple outputs, and Loops helps users recognize and understand these changes. Second, tracking provenance remains important to understand the evolution of analysis or comparing alternatives. However, reactive notebooks do not have to follow the same structure as sequentially executed notebooks (see Sec. 3). Automatic re-evaluation of cells after a change can generate unintended versions of the notebook.

8 Conclusion

Analysts face challenges and pain points when working with notebooks [9]. With the Loops approach described in this paper, we address the challenges related to reproducing and comparing notebooks. Loops tracks the provenance of notebooks to assist analysts during their work. We visualize the evolution of the notebook over time and highlight differences between versions. We have carefully designed visualizations that consistently visualize changes for the whole notebook and various types of content present in notebooks. This gives analysts direct feedback on their changes and supports them at various stages of the data science process. Based on the feedback from four notebook users, we are confident that Loops can effectively support data science processes in notebooks.

Supplemental Materials

All supplemental materials are available on OSF at osf.io/79eyn. In particular, they include (1) a full version of this paper with all appendices, (2) a video to demonstrate the workflow and interactions in Loops, and (3) the source code of our prototype together with the datasets used for the uses cases and all figures of our presented approach.

Acknowledgments

We thank Raisa Romanov Geleta, Oleg Lesota, Rubina Waldbauer, and Mohammed Abbass for their participation in the interviews and their valuable feedback. We also thank Kai Xu for his feedback on Loops.

This work was supported by the Austrian Science Fund (FWF DFH 23–N), and the Austrian Research Promotion Agency (FFG 881844). Pro2Future is funded within the Austrian COMET Program under the auspices of the Austrian Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology, the Austrian Federal Ministry for Digital and Economic Affairs, and of the States of Upper Austria and Styria. COMET is managed by the Austrian Research Promotion Agency FFG.

The authors acknowledge the American Association for Cancer Research and its financial and material support in the development of the AACR Project GENIE registry, and members of the consortium for their commitment to open data. Interpretations are the responsibility of the authors.

References

[1] AACR Project Genie Consortium. AACR Project GENIE: powering precision medicine through an international consortium. *Cancer discovery*, 7(8):818–831, 2017. doi: 10/ggnnrn 7

[2] J. Abello, S. Hadlak, H. Schumann, and H.-J. Schulz. A Modular Degree-of-Interest Specification for the Visual Analysis of Large Dynamic Networks. *IEEE Transactions on Visualization and Computer Graphics*, 20(3):337–350, 2014. doi: 10.1109/TVCG.2013.109 9

[3] S. Alspaugh, N. Zokaei, A. Liu, C. Jin, and M. A. Hearst. Futzing and Moseying: Interviews with Professional Data Analysts on Exploration Practices. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):22–31, 2019. doi: 10.1109/TVCG.2018.2865040 1, 2, 5, 8

[4] B. Bach, C. Shi, N. Heulot, T. Madhyastha, T. Grabowski, and P. Dragicevic. Time Curves: Folding Time to Visualize Patterns of Temporal Evolution in Data. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):559–568, 2016. doi: 10.1109/TVCG.2015.2467851 3

[5] F. Bolte and S. Bruckner. Vis-a-Vis: Visual Exploration of Visualization Source Code Evolution. *IEEE Transactions on Visualization and Computer Graphics*, 27(7):3153–3167, 2021. doi: 10.1109/TVCG.2019.2963651 3

[6] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011. doi: 10.1109/TVCG.2011.185 7

[7] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 7, 14

[8] W. Burger and M. J. Burge. *Digital Image Processing: An Algorithmic Introduction Using Java*. Texts in computer science. Springer, 1 ed., 2008. 6

[9] S. Chattopadhyay, I. Prasad, A. Z. Henley, A. Sarma, and T. 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*, pp. 1–12. ACM, New York, USA, 2020. doi: 10.1145/3313831.3376729 2, 3, 4, 5, 8, 9

[10] A. Crisan, B. Fiore-Gartland, and M. Tory. Passing the Data Baton : A Retrospective Analysis on Data Science Work and Workers. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1860–1870, 2021. doi: 10.1109/TVCG.2020.3030340 1

[11] Z. T. Cutler, K. Gadhave, and A. Lex. Trrack: A Library for Provenance Tracking in Web-Based Visualizations. In *IEEE Visualization Conference (VIS)*, pp. 116–120, 2020. doi: 10.1109/VIS47514.2020.00030 7

[12] K. Eckelt, P. Adelberger, M. J. Bauer, T. Zichner, and M. Streit. Kokiri: Random-Forest-Based Comparison and Characterization of Cohorts. *IEEE VIS Workshop on Visualization in Biomedical AI*, 2022. doi: 10.1101/2022.08.16.503622 7

[13] K. Eckelt, A. Hinterreiter, P. Adelberger, C. Walchshofer, V. Dhanoa, C. Humer, M. Heckmann, C. Steinparz, and M. Streit. Visual Exploration of Relationships and Structure in Low-Dimensional Embeddings. *IEEE Transactions on Visualization and Computer Graphics*, 29(7):3312–3326, 2023. doi: 10.1109/TVCG.2022.3156760 3

[14] W. Epperson, V. Gorantla, D. Moritz, and A. Perer. Dead or Alive: Continuous Data Profiling for Interactive Data Science. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–11, 2023. doi: 10.1109/TVCG.2023.3327367 2

[15] W. Epperson, D. Jung-Lin Lee, L. Wang, K. Agarwal, A. G. Parameswaran, D. Moritz, and A. Perer. Leveraging Analysis History for Improved In Situ Visualization Recommendation. *Computer Graphics Forum*, 41(3):145–155, 2022. doi: 10.1111/cgf.14529 2

[16] R. Faust, C. Scheidegger, and C. North. Aardvark: Comparative visualization of data analysis scripts. In *2023 IEEE Visualization in Data Science (VDS)*, pp. 30–38, 2023. doi: 10.1109/VDS60365.2023.00009 2

[17] K. Gadhave, Z. Cutler, and A. Lex. Persist: Persistent and reusable interactions in computational notebooks. *OSF Preprint*, 2024. doi: 10.31219/osf.io/9x8eq 2

[18] M. Gleicher. Considerations for Visualizing Comparison. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):413–423, 2018. doi: 10.1109/TVCG.2017.2744199 3, 6

[19] M. Gleicher, D. Albers, R. Walker, I. Jusufi, C. D. Hansen, and J. C. Roberts. Visual comparison for information visualization. *Information Visualization*, 10(4):289 –309, 2011. doi: 10.1177/1473871611416549 3, 7

[20] R. Gonzalez and R. Woods. *Digital Image Processing, Global Edition*. Pearson Education, 2018. 3, 6

[21] S. Gratzl, A. Lex, N. Gehlenborg, N. Cosgrove, and M. Streit. From Visual Exploration to Storytelling and Back Again. *Computer Graphics Forum*, 35(3):491–500, 2016. doi: 10.1111/cgf.12925 2

[22] A. 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/`, 2020. Accessed: 2024-03-31. 1, 2

[23] A. Head, F. Hohman, T. Barik, S. M. Drucker, and R. DeLine. Managing Messes in Computational Notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–12. ACM, New York, USA, 2019. doi: 10.1145/3290605.3300500 1, 2

[24] J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala. Graphical Histories for Visualization: Supporting Analysis, Communication, and Evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1189–1196, 2008. doi: 10.1109/TVCG.2008.137 2

[25] F. Hohman, K. Wongsuphasawat, M. B. Kery, and K. Patel. Understanding and Visualizing Data Iteration in Machine Learning. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pp. 1–13. ACM, New York, USA, 2020. doi: 10.1145/3313831.3376177 1, 3, 5

[26] B. V. Hollingsworth, S. E. Reichenbach, Q. Tao, and A. Visvanathan. Comparative visualization for comprehensive two-dimensional gas chromatography. *Journal of Chromatography A*, 1105(1):51–58, 2006. doi: 10.1016/j.chroma.2005.11.074 3

[27] ImageMagick Studio LLC. Comparing – ImageMagick Examples. `https://imagemagick.org/Usage/compare/`, 2012. Accessed: 2024-03-31. 3

[28] N. Jardine, B. D. Ondov, N. Elmqvist, and S. Franconeri. The Perceptual Proxies of Visual Comparison. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1012–1021, 2020. doi: 10.1109/TVCG.2019.2934786 3, 7

[29] Jupyter Development Team. nbdime – diffing and merging of jupyter notebooks — nbdime 4.0.0a1 documentation. `https://nbdime.readthedocs.io/`, 2023. Accessed: 2024-03-31. 2

[30] Jupyter Development Team. jupyterlab-git. `https://github.com/jupyterlab/jupyterlab-git`, 2024. Accessed: 2024-03-31. 2

[31] M. B. Kery, B. E. John, P. O'Flaherty, A. Horvath, and B. A. Myers. Towards Effective Foraging by Data Scientists to Find Past Analysis Choices. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–13. ACM, New York, USA, 2019. doi: 10.1145/3290605.3300322 2

[32] M. B. Kery and B. A. Myers. Interactions for Untangling Messy History in a Computational Notebook. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 147–155, 2018. doi: 10.1109/VLHCC.2018.8506576 4

[33] D. Kerzel, S. Samuel, and B. König-Ries. Towards Tracking Provenance from Machine Learning Notebooks. In *Proceedings of the 13th International Joint Conference on Knowledge Discovery, Knowledge Engineering*

and Knowledge Management - KDIR,, pp. 274–281. SciTePress, 2021. doi: 10.5220/0010681400003064 2

[34] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and Jupyter Development Team. Jupyter Notebooks—a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pp. 87–90. IOS Press, 2016. doi: 10.3233/978-1-61499-649-1-87 1

[35] D. E. Knuth. Literate Programming. *The Computer Journal*, 27(2):97–111, 1984. doi: 10.1093/comjnl/27.2.97 1

[36] R. Kosara. Notebooks for Data Analysis and Visualization: Moving Beyond the Data. *IEEE Computer Graphics and Applications*, 43(1):91–96, 2023. doi: 10.1109/MCG.2022.3222024 7, 9

[37] S. LYi, J. Jo, and J. Seo. Comparative Layouts Revisited: Design Space, Guidelines, and Future Directions. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1525–1535, Feb. 2021. Conference Name: IEEE Transactions on Visualization and Computer Graphics. doi: 10.1109/TVCG.2020.3030419 7

[38] M. M. Malik, C. Heinzl, and M. E. Groeller. Comparative Visualization for Parameter Studies of Dataset Series. *IEEE Transactions on Visualization and Computer Graphics*, 16(5):829–840, 2010. doi: 10.1109/TVCG.2010.20 3

[39] Microsoft. Monaco editor - a browser based code editor. `https://github.com/microsoft/monaco-editor`, 2024. Accessed: 2024-03-31. 7

[40] M. Muller, I. Lange, D. Wang, D. Piorkowski, J. Tsay, Q. V. Liao, C. Dugan, and T. Erickson. How data science workers work with data: Discovery, capture, curation, design, creation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 15 pages, p. 1–15. ACM, New York, USA, 2019. doi: 10.1145/3290605.3300356 5

[41] E. W. Myers. AnO(ND) difference algorithm and its variations. *Algorithmica*, 1(1):251–266, 1986. doi: 10.1007/BF01840446 3, 5

[42] A. H. Nassar, E. Adib, and D. J. Kwiatkowski. Distribution of KRASG12C Somatic Mutations across Race, Sex, and Cancer Type. *New England Journal of Medicine*, 384(2):185–187, 2021. doi: 10/ghvp8t 7

[43] C. Niederer, H. Stitz, R. Hourieh, F. Grassinger, W. Aigner, and M. Streit. TACO: Visualizing Changes in Tables Over Time. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):677–686, 2017. doi: 10.1109/TVCG.2017.2745298 3, 5, 9

[44] Y. S. Nugroho, H. Hata, and K. Matsumoto. How different are different diff algorithms in Git? *Empirical Software Engineering*, 25(1):790–823, 2020. doi: 10.1007/s10664-019-09772-z 3, 5

[45] B. Ondov, N. Jardine, N. Elmqvist, and S. Franconeri. Face to Face: Evaluating Visual Comparison. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):861–871, 2019. doi: 10.1109/TVCG.2018.2864884 3, 7

[46] H.-G. Pagendarm and F. Post. Comparative visualization - approaches and examples. In *5th Eurographics Workshop on Visualization in Scientific Computing Rostock*, 1994. 9

[47] J. M. Perkel. Reactive, reproducible, collaborative: computational notebooks evolve. *Nature*, 593(7857):156–157, 2021. doi: 10.1038/d41586-021-01174-w 1, 9

[48] H. Pflüger, D. Thom, A. Schütz, D. Bohde, and T. Ertl. VeCHArt: Visually Enhanced Comparison of Historic Art Using an Automated Line-Based Synchronization Technique. *IEEE Transactions on Visualization and Computer Graphics*, 26(10):3063–3076, Oct. 2020. doi: 10.1109/TVCG.2019.2908166 9

[49] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire. A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pp. 507–517, 2019. doi: 10.1109/MSR.2019.00077 1, 2

[50] P. Pirolli and S. Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of International Conference on Intelligence Analysis*, vol. 5, pp. 2–4. McLean, VA, USA, 2005. 1

[51] F. Psallidas, Y. Zhu, B. Karlas, J. Henkel, M. Interlandi, S. Krishnan, B. Kroth, V. Emani, W. Wu, C. Zhang, M. Weimer, A. Floratou, C. Curino, and K. Karanasos. Data Science Through the Looking Glass: Analysis of Millions of GitHub Notebooks and ML.NET Pipelines. *ACM SIGMOD Record*, 51(2):30–37, 2022. doi: 10.1145/3552490.3552496 1

[52] D. Ramasamy, C. Sarasua, A. Bacchelli, and A. Bernstein. Visualising data science workflows to support third-party notebook comprehension: an empirical study. *Empirical Software Engineering*, 28(3):58, 2023. doi: 10.1007/s10664-023-10289-9 2, 4

[53] Resemble.js. Image Analysis and Comparison. `https://github.com/rsmbl/Resemble.js`, 2023. Accessed: 2024-03-31. 3

[54] A. Rule, A. Tabard, and J. D. Hollan. Exploration and Explanation in Computational Notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pp. 1–12. ACM, Montreal, Canada, 2018. doi: 10.1145/3173574.3173606 2, 4, 5, 7, 8

[55] S. Samuel and B. König-Ries. ProvBook: Provenance-based Semantic Enrichment of Interactive Notebooks for Reproducibility. In *International Semantic Web Conference (ISWC) Demo Track 2018*, 2018. 2

[56] J. Schmidt, M. E. Gröller, and S. Bruckner. VAICo: Visual Analysis for Image Comparison. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2090–2099, 2013. doi: 10.1109/TVCG.2013.213 3

[57] H. Stitz, S. Gratzl, H. Piringer, T. Zichner, and M. Streit. KnowledgePearls: Provenance-Based Visualization Retrieval. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):120–130, 2019. doi: 10.1109/TVCG.2018.2865024 2

[58] C. Studholme, D. L. G. Hill, and D. J. Hawkes. An overlap invariant entropy measure of 3D medical image alignment. *Pattern Recognition*, 32(1):71–86, 1999. doi: 10.1016/S0031-3203(98)00091-0 7, 14

[59] A. Tang. @armantang/html-diff - generate html content diffs. `https://github.com/Arman19941113/html-diff`, 2023. Accessed: 2024-03-31. 7

[60] J. D. Team. Jupyter notebook 2015 ux survey results. `https://github.com/jupyter/surveys/blob/master/surveys/2015-12-notebook-ux/analysis/report_dashboard.ipynb`, 2018. (Accessed: 2024-03-31). 2

[61] Q. D. Team. Quarto. `https://quarto.org/`, 2024. (Accessed: 2024-03-31). 2

[62] A. Y. Wang, W. Epperson, R. A. DeLine, and S. M. Drucker. Diff in the Loop: Supporting Data Comparison in Exploratory Data Analysis. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pp. 1–10. ACM, New York, USA, 2022. doi: 10.1145/3491102.3502123 3

[63] A. Y. Wang, A. Mittal, C. Brooks, and S. Oney. How Data Scientists Use Computational Notebooks for Real-Time Collaboration. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):39:1–39:30, Nov. 2019. doi: 10.1145/3359141 9

[64] J. Wang, T.-Y. KUO, L. Li, and A. Zeller. Assessing and Restoring Reproducibility of Jupyter Notebooks. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 138–149, 2020. 1

[65] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861 7, 14

[66] N. Weinman, S. M. Drucker, T. Barik, and R. DeLine. Fork It: Supporting Stateful Alternatives in Computational Notebooks. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM, New York, USA, 2021. doi: 10.1145/3411764.3445527 2

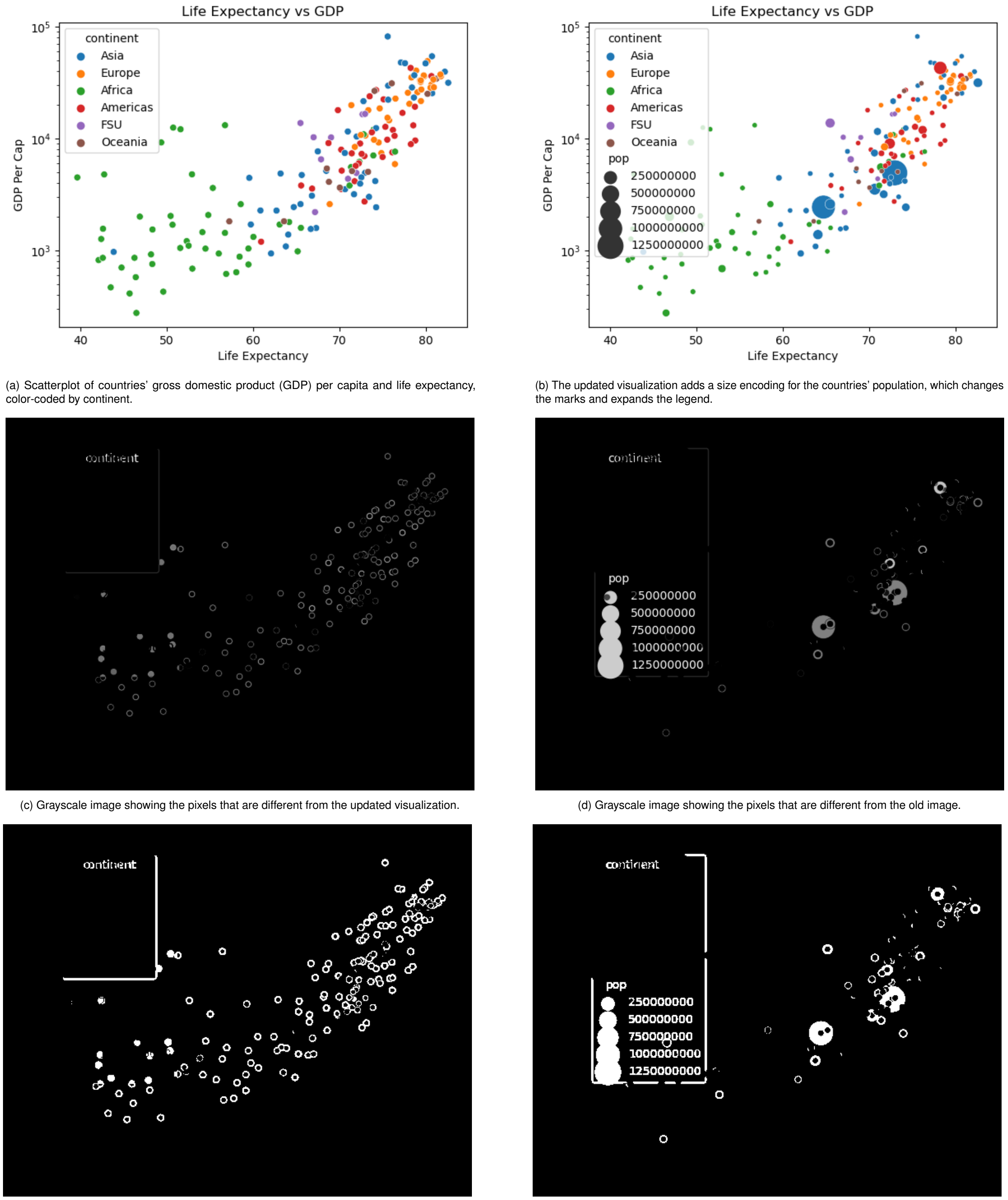
[67] J. Wenskovitch, J. Zhao, S. Carter, M. Cooper, and C. North. Albireo: An Interactive Tool for Visually Summarizing Computational Notebook Structure. In *2019 IEEE Visualization in Data Science (VDS)*, pp. 1–10, 2019. doi: 10.1109/VDS48975.2019.8973385 1

A Image Difference Visualization

A.1 Calculating Image Differences

The following figures illustrate the steps of our image difference calculation using the visualizations shown in Fig. 15.

Fig. 7 shows the two compared versions of the visualization and how the differences are determined. Fig. 8 shows the morphological operations on the differences and the color-coding of the differences. The colored differences are then blended with the visualizations and presented juxtaposed—as in Fig. 15—or superimposed.



(a) Scatterplot of countries' gross domestic product (GDP) per capita and life expectancy, color-coded by continent.

(b) The updated visualization adds a size encoding for the countries' population, which changes the marks and expands the legend.

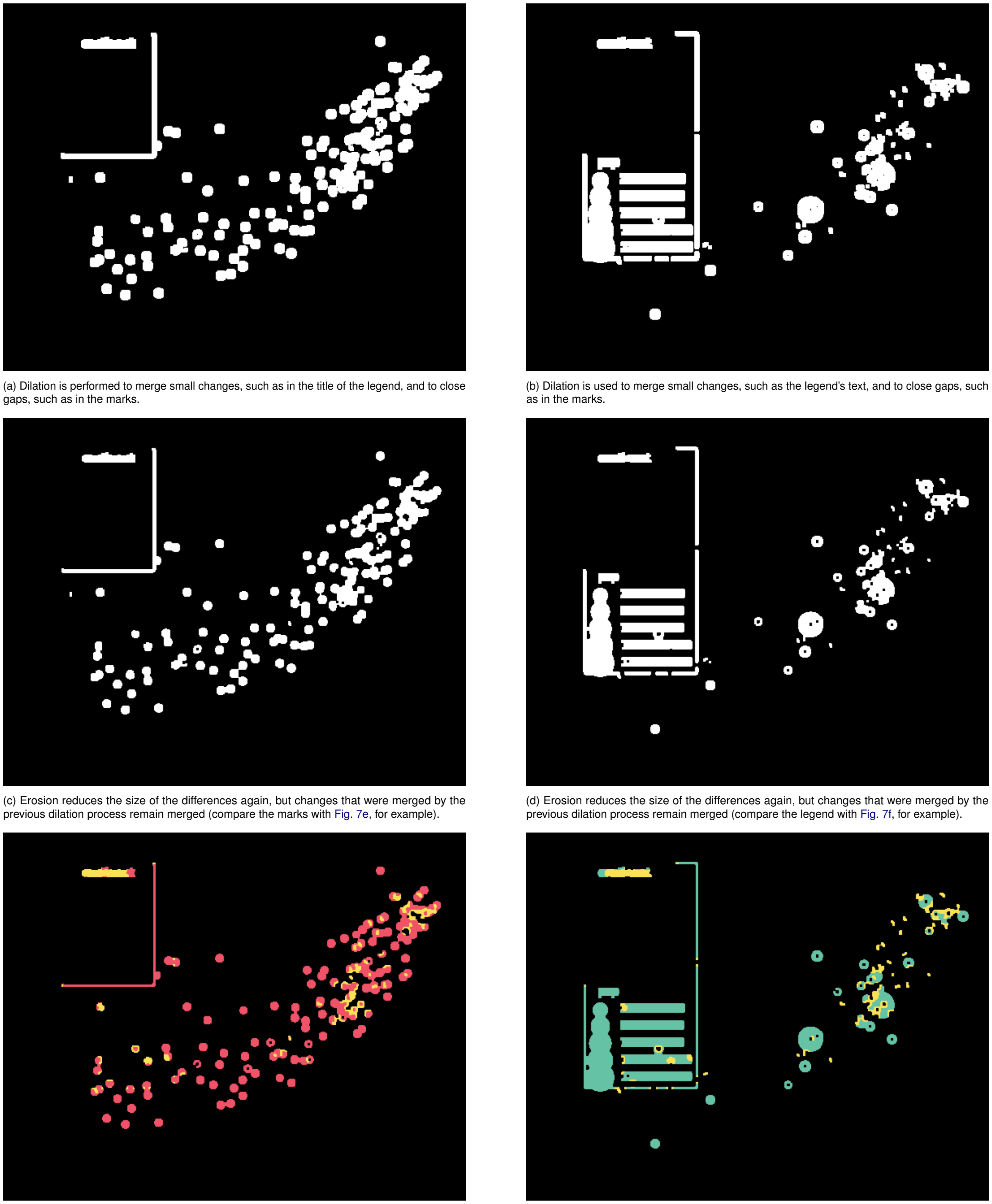
(c) Grayscale image showing the pixels that are different from the updated visualization.

(d) Grayscale image showing the pixels that are different from the old image.

(e) The grayscale image is converted to a binary image by thresholding the differences. The image shows which of the marks became smaller due to the size encoding and that the legend became longer and wider, removing the previous bottom and right borders.

(f) The grayscale image is converted to a binary image by thresholding the differences. The image shows which of the marks came larger due to the size encoding and the new legend entries for the size encoding.

Fig. 7: Calculation of differences between two version of a scatterplot. To determine the differences between the two versions, the images are first subtracted from each other (c and d), and then converted into a binary image by tresholding (e and f). We then apply morphological operations to the binary images as shown in Fig. 8.



(a) Dilation is performed to merge small changes, such as in the title of the legend, and to close gaps, such as in the marks.

(b) Dilation is used to merge small changes, such as the legend's text, and to close gaps, such as in the marks.

(c) Erosion reduces the size of the differences again, but changes that were merged by the previous dilation process remain merged (compare the marks with Fig. 7e, for example).

(d) Erosion reduces the size of the differences again, but changes that were merged by the previous dilation process remain merged (compare the legend with Fig. 7f, for example).

(e) The differences in Fig. 8c are compared with those in Fig. 8d. Differences (white pixels) that do not occur in the other are color-coded red (removed), while differences that occur in both images are color-coded yellow (changed).

(f) The differences inFig. 8d are compared with those in Fig. 8c. Differences (white pixels) that do not occur in the other are color-coded green (added), while differences that occur in both images are color-coded yellow (changed).

Fig. 8: After calculating the differences between the two images as shown in Fig. 7, we apply morphological operations on the images. We first dilate the binary difference images to close gaps and merge adjacent changes (a and b). Dilation is performed twice and in the next step (c and d) the difference images are eroded again. Erosion does not completely reverse dilatation, as, for example, gaps that have been closed by dilatation are not reopened to their original size. In the last step (e and f) the images are color-coded, with differences that are only contained in one difference image being shown in red or green, respectively, while parts that are different in both images are shown in yellow. Since we have applied the dilation twice, but the erosion only once, the color-coded pixels are slightly larger than the differences and envelope them (see Fig. 15).

A.2 Image Similarity Metrics

We considered multiple image similarity metrics to estimate the amount of change between pairs of images: (*i*) Mean Squared Error, (*ii*) Structured Similarity Index [65], (*iii*) Normalized Mututal Information [58], and (*iv*) Oriented FAST and Rotated BRIEF (ORB) [7].

We excluded the Mean Squared Error from further consideration, as its value is unbound and therefore hard to interpret. The Normalized Mutual Information compares intensities of corresponding pixels in two images and determines whether the images are uncorrelated or correlated [58]. As there is no differentiation between positive or negative correlation we also excluded the NMI from further considerations. The Structured Similarity Index (SSIM) measures image quality by quantifying the degradation of structural information [65], which is information from neighboring pixels. Our assumption is that this could be applied to visualizations when marks change position and thus their structure. Oriented FAST and Rotated BRIEF (ORB) is a feature detector, which we apply to detect and match features in the image pairs and subsequently quantify how well they match as a similarity metric. We consider matching features between two data visualizations promising for cases where, for example, the legend changes position but not its content.

In Tab. 1, we provide the results for SSIM, ORB, and our relative pixel similarity for the various image difference figures we present in this paper. Overall, the results of SSIM, ORB, and relative pixel similarity are very similar overall. We decided to use the relative pixel similarity as it is computed directly from the explicit difference encoding, ensuring that the visualized differences match the calculated similarity.

Table 1: Comparison of the SSIM and ORB similarity metrics and the relative pixel similarity we use for the image difference visualizations presented in this paper.

	Similarity Score for								
Metric	**Fig. 6**	**Fig. 9**	**Fig. 10**	**Fig. 11**	**Fig. 13**	**Fig. 14**	**Fig. 15**	**Fig. 16**	**Fig. 12**
SSIM	97.9%	99.9%	99.4%	91.9%	92.1%	95.1%	96.8%	99.5%	100%
ORB	92.6%	99.9%	98.6%	85.3%	86.1%	90.0%	92.2%	99.4%	100%
Relative Pixel Similarity	89.3%	99.7%	99.0%	86.3%	84.0%	94.6%	89.8%	97.6%	100%

A.3 Examples

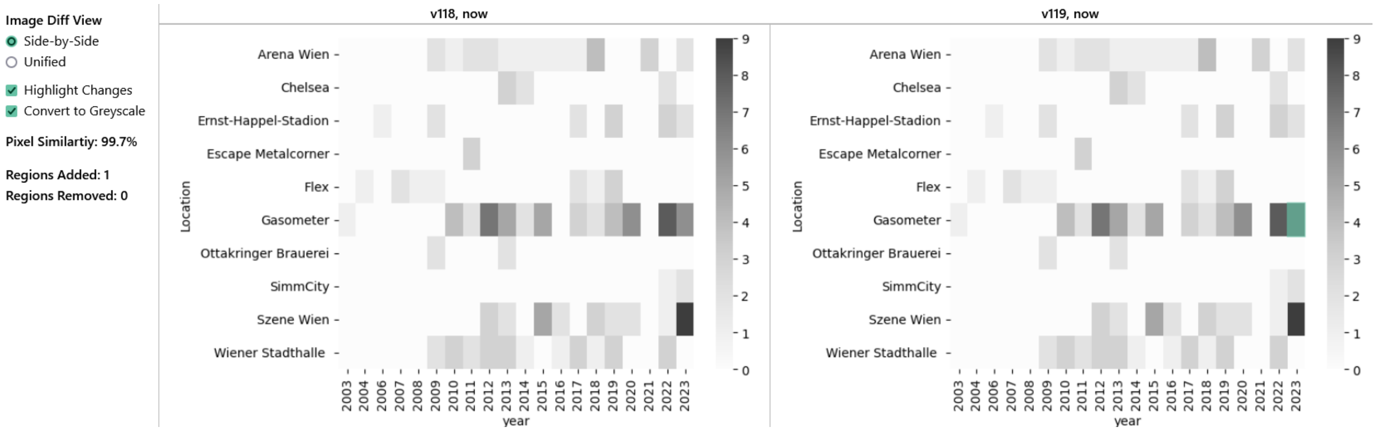


Image Diff View

● Side-by-Side

○ Unified

☑ Highlight Changes

☑ Convert to Greyscale

Pixel Similartiy: 99.7%

Regions Added: 1

Regions Removed: 0

Fig. 9: Heatmap with the concert data from Use Case 1, showing the number of concerts per year in Viennese locations. For the difference visualization, the heatmap is shown in grayscale, such that only the explicit encoding of differences is colored. The change is clearly highlighted on top of the new heatmap on the right, showing that between the two versions, the data of 2023 changed, adding additional concerts in the *Gasometer* location.

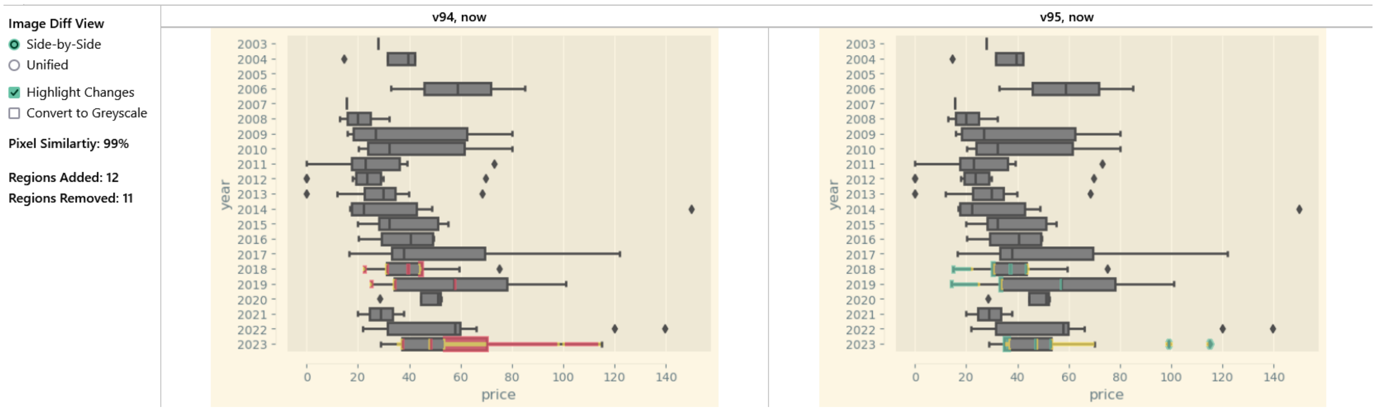


Image Diff View

● Side-by-Side

○ Unified

☑ Highlight Changes

□ Convert to Greyscale

Pixel Similartiy: 99%

Regions Added: 12

Regions Removed: 11

Fig. 10: Box plot with the concert data from Use Case 1, showing the distribution of ticket prices per year. This box plot uses the *Solarized* theme. We have disabled the conversion to grayscale to highlight how our difference visualization adapts to non-white background colors. The explicit difference encoding shows that the length of the lower whiskers was increased for the boxplots of 2018 and 2019 by highlighting the new pixels. For 2023, the box plots upper whisker and quartile shrunk significantly, shown by the removed pixels highlighted in red.

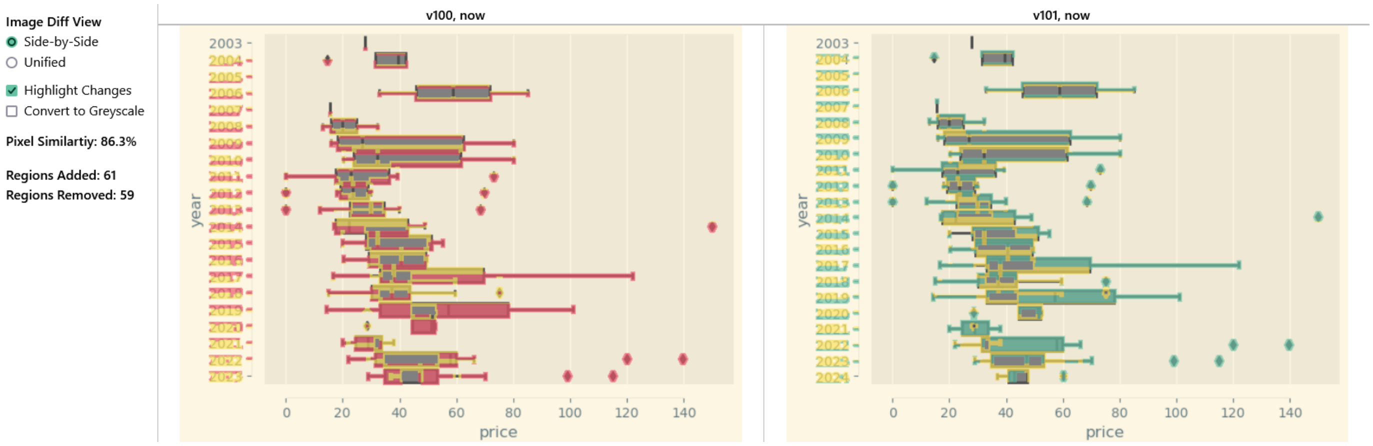


Image Diff View

● Side-by-Side

○ Unified

☑ Highlight Changes

□ Convert to Greyscale

Pixel Similartiy: 86.3%

Regions Added: 61

Regions Removed: 59

Fig. 11: Box plot with the concert data from Use Case 1, showing the distribution of ticket prices per year. This box plot uses the *Solarized* theme. We have disabled the conversion to grayscale to highlight how our difference visualization adapts to non-white background colors. In the new version, data for 2024 became available, resulting in a denser distribution of labels to accommodate the expanded timeframe. The difference visualization shows that except for the first year, all boxplots and year labels move slightly upwards.

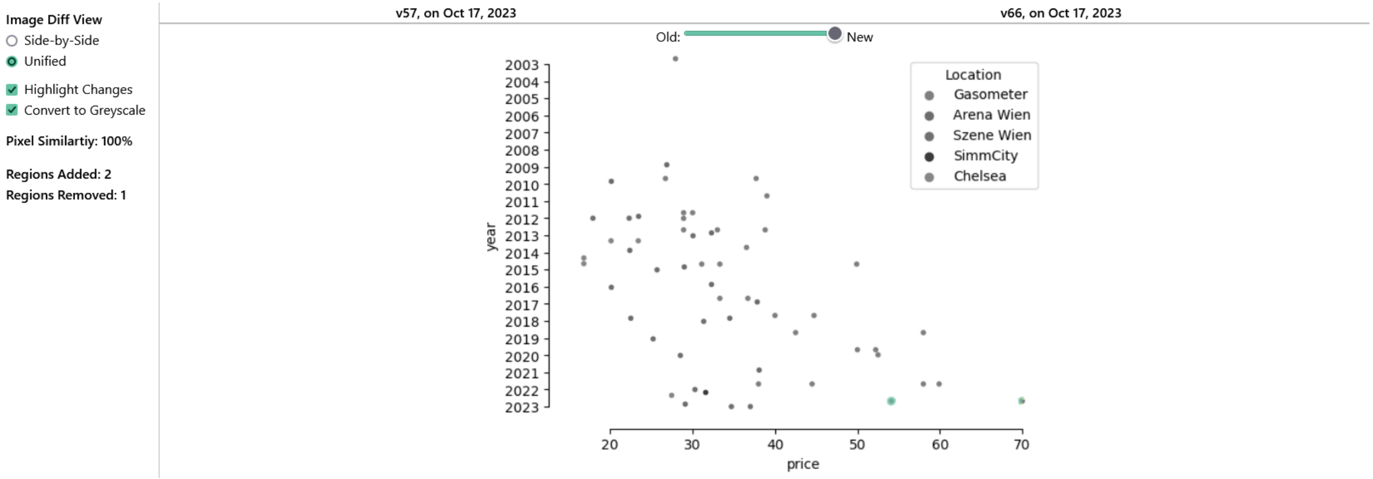


Image Diff View

○ Side-by-Side

● Unified

☑ Highlight Changes

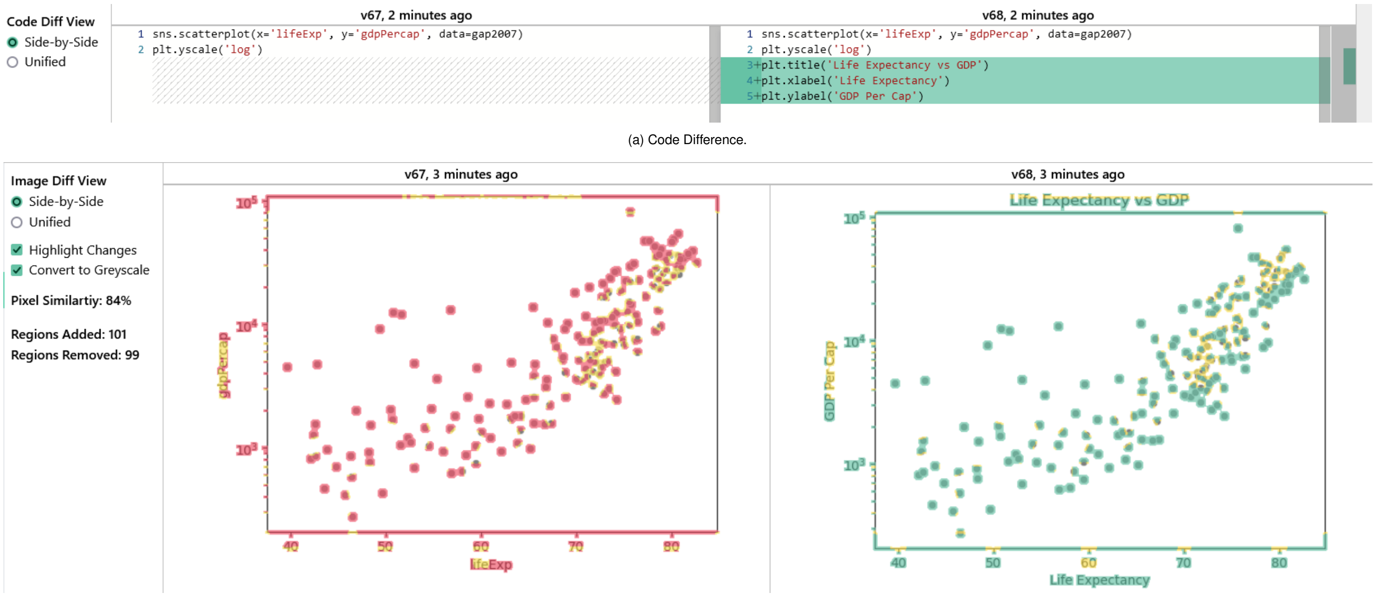
☑ Convert to Greyscale

Pixel Similartiy: 100%

Regions Added: 2

Regions Removed: 1

Fig. 12: Scatterplot with the concert data from Use Case 1, showing the concerts' ticket price and year. In the new version, one new point is added to the scatterplot as the data were updated. The difference visualization highlights the point in green and by varying the opacity of the images with the slider the change between the two versions becomes even more apparent.



v67, 2 minutes ago

```
sns.scatterplot(x='lifeExp', y='gdpPercap', data=gap2007)
plt.yscale('log')
```

v68, 2 minutes ago

```
sns.scatterplot(x='lifeExp', y='gdpPercap', data=gap2007)
plt.yscale('log')
+plt.title('Life Expectancy vs GDP')
+plt.xlabel('Life Expectancy')
+plt.ylabel('GDP Per Cap')
```

(a) Code Difference.

Pixel Similartiy: 84%

Regions Added: 101

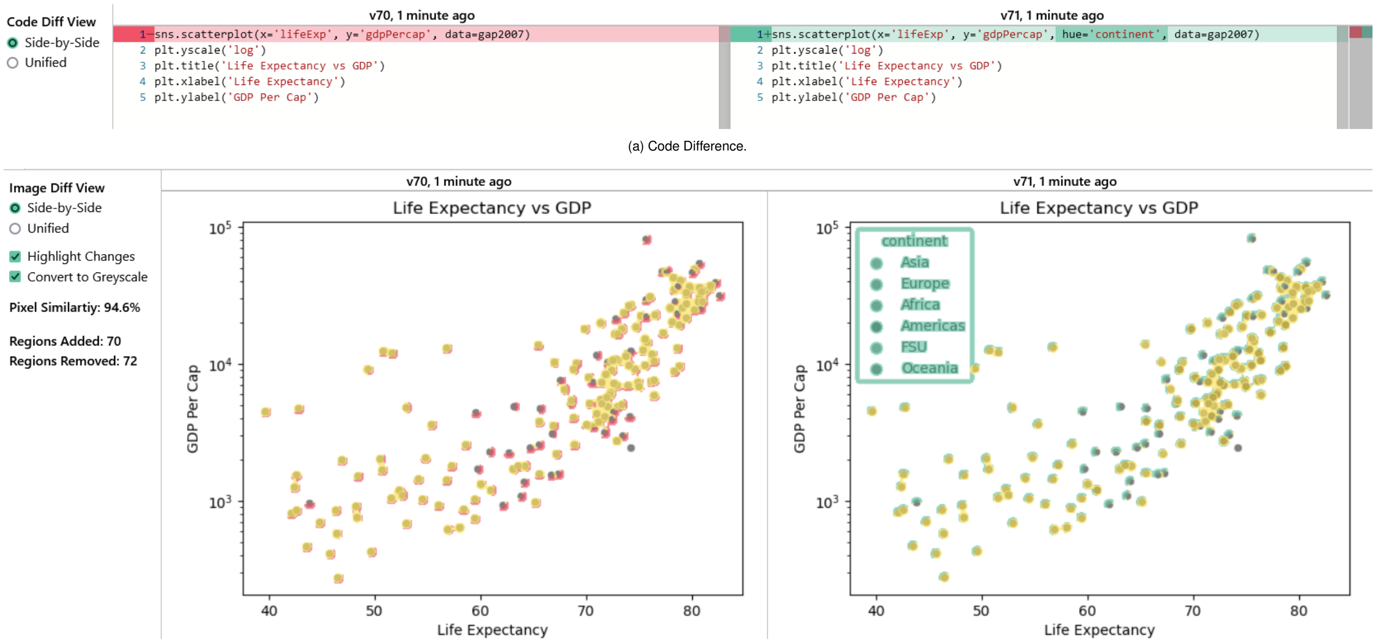
Regions Removed: 99

v67, 3 minutes ago

v68, 3 minutes ago

(b) Visualization Difference

Fig. 13: Scatterplot of gross domestic product (GDP) per capita and life expectancy. The new version adds a title for the figure and labels for the axes, causing the visualization to increase in size and its content to move.



v70, 1 minute ago

```
-sns.scatterplot(x='lifeExp', y='gdpPercap', data=gap2007)
plt.yscale('log')
plt.title('Life Expectancy vs GDP')
plt.xlabel('Life Expectancy')
plt.ylabel('GDP Per Cap')
```

v71, 1 minute ago

```
+sns.scatterplot(x='lifeExp', y='gdpPercap', hue='continent', data=gap2007)
plt.yscale('log')
plt.title('Life Expectancy vs GDP')
plt.xlabel('Life Expectancy')
plt.ylabel('GDP Per Cap')
```

(a) Code Difference.

Pixel Similartiy: 94.6%

Regions Added: 70

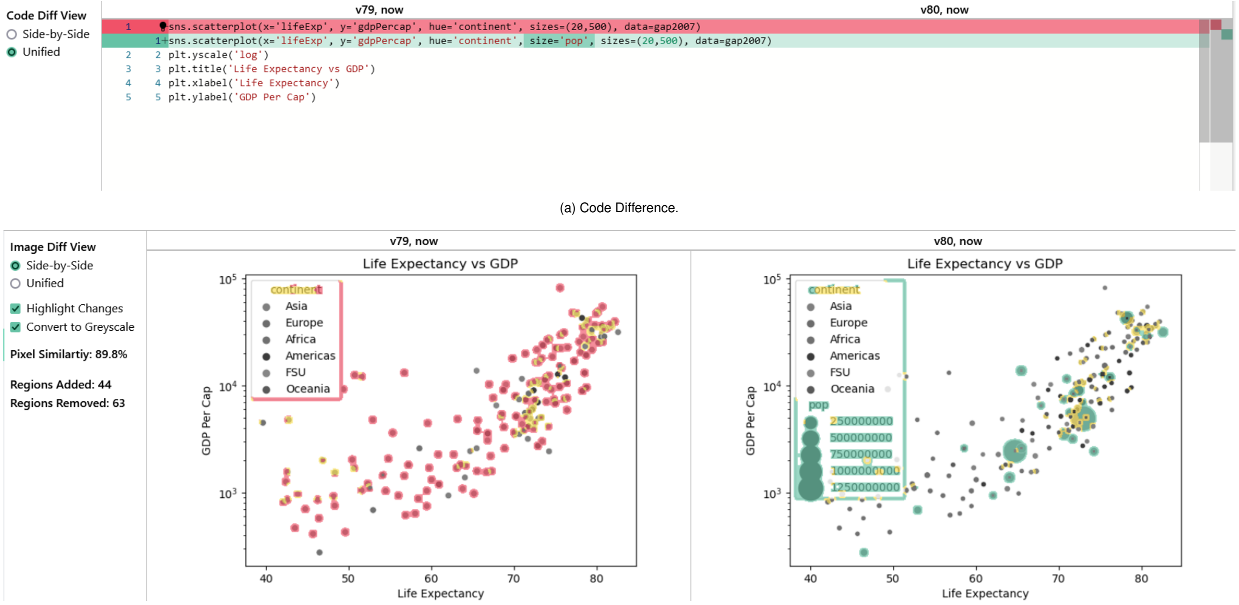
Regions Removed: 72

v70, 1 minute ago

v71, 1 minute ago

(b) Visualization Difference

Fig. 14: Scatterplot of gross domestic product (GDP) per capita and life expectancy. The new version adds a color encoding, resulting in changes to the individual marks and the addition of a legend.



Code Diff View
○ Side-by-Side
● Unified

v79, now

v80, now

```
sns.scatterplot(x='lifeExp', y='gdpPercap', hue='continent', sizes=(20,500), data=gap2007)
+sns.scatterplot(x='lifeExp', y='gdpPercap', hue='continent', size='pop', sizes=(20,500), data=gap2007)
plt.yscale('log')
plt.title('Life Expectancy vs GDP')
plt.xlabel('Life Expectancy')
plt.ylabel('GDP Per Cap')
```

(a) Code Difference.

Image Diff View
● Side-by-Side
○ Unified

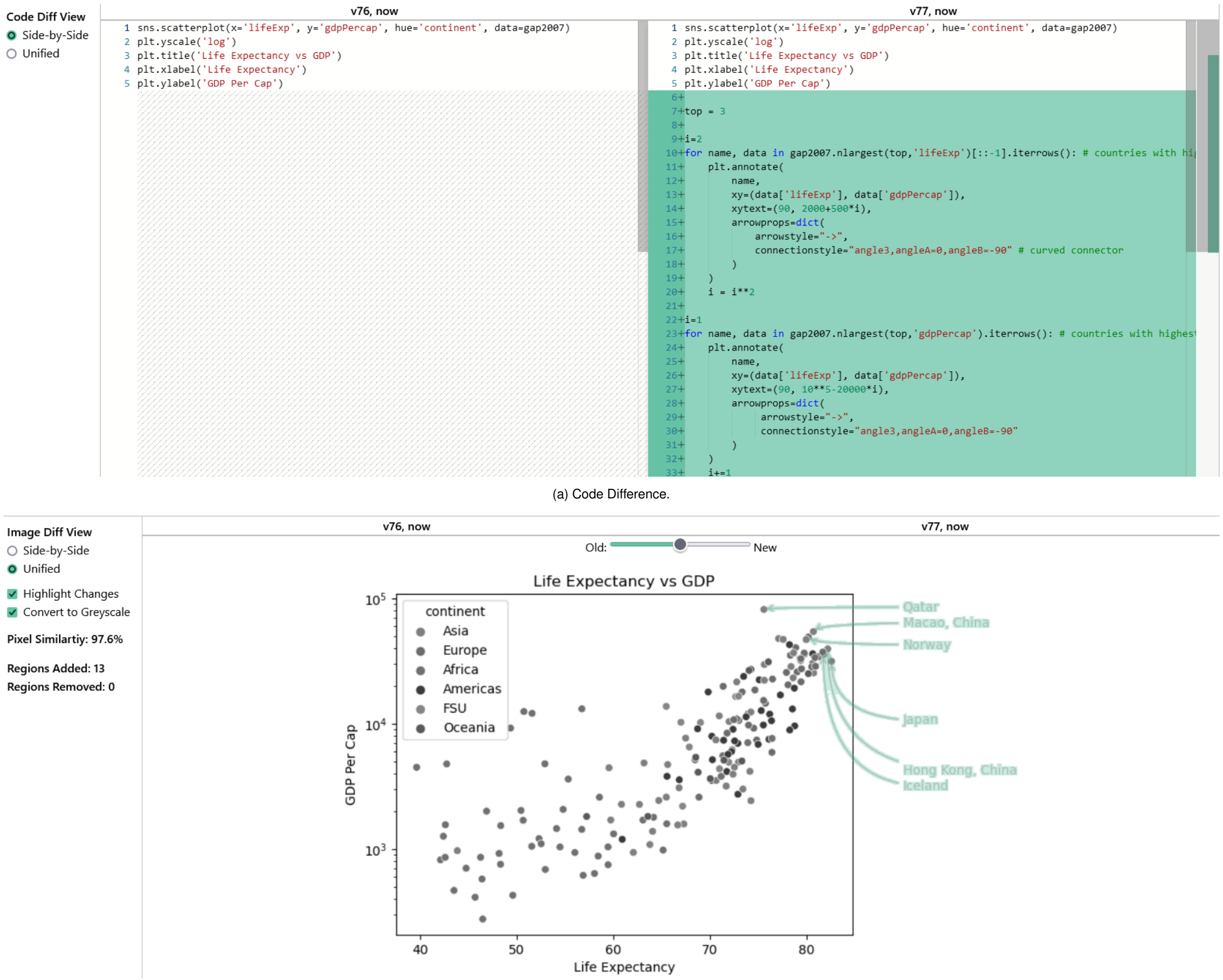
☑ Highlight Changes
☑ Convert to Greyscale

Pixel Similartiy: 89.8%

Regions Added: 44
Regions Removed: 63

(b) Visualization Difference

Fig. 15: Scatterplot of gross domestic product (GDP) per capita and life expectancy. The new version adds a size encoding, resulting in an extended legend and marks getting larger (added pixels in green) or smaller (removed pixels in red).



Code Diff View
● Side-by-Side
○ Unified

v76, now

```
sns.scatterplot(x='lifeExp', y='gdpPercap', hue='continent', data=gap2007)
plt.yscale('log')
plt.title('Life Expectancy vs GDP')
plt.xlabel('Life Expectancy')
plt.ylabel('GDP Per Cap')
```

v77, now

```
sns.scatterplot(x='lifeExp', y='gdpPercap', hue='continent', data=gap2007)
plt.yscale('log')
plt.title('Life Expectancy vs GDP')
plt.xlabel('Life Expectancy')
plt.ylabel('GDP Per Cap')
+
+top = 3
+
+i=2
+for name, data in gap2007.nlargest(top,'lifeExp')[::-1].iterrows(): # countries with hi
+    plt.annotate(
+        name,
+        xy=(data['lifeExp'], data['gdpPercap']),
+        xytext=(90, 2000+500*i),
+        arrowprops=dict(
+            arrowstyle="->",
+            connectionstyle="angle3,angleA=0,angleB=-90" # curved connector
+        )
+    )
+    i = i**2
+
+i=1
+for name, data in gap2007.nlargest(top,'gdpPercap').iterrows(): # countries with highest
+    plt.annotate(
+        name,
+        xy=(data['lifeExp'], data['gdpPercap']),
+        xytext=(90, 10**5-20000*i),
+        arrowprops=dict(
+             arrowstyle="->",
+             connectionstyle="angle3,angleA=0,angleB=-90"
+        )
+    )
+    i+=1
```

(a) Code Difference.

Image Diff View
○ Side-by-Side
● Unified

☑ Highlight Changes
☑ Convert to Greyscale

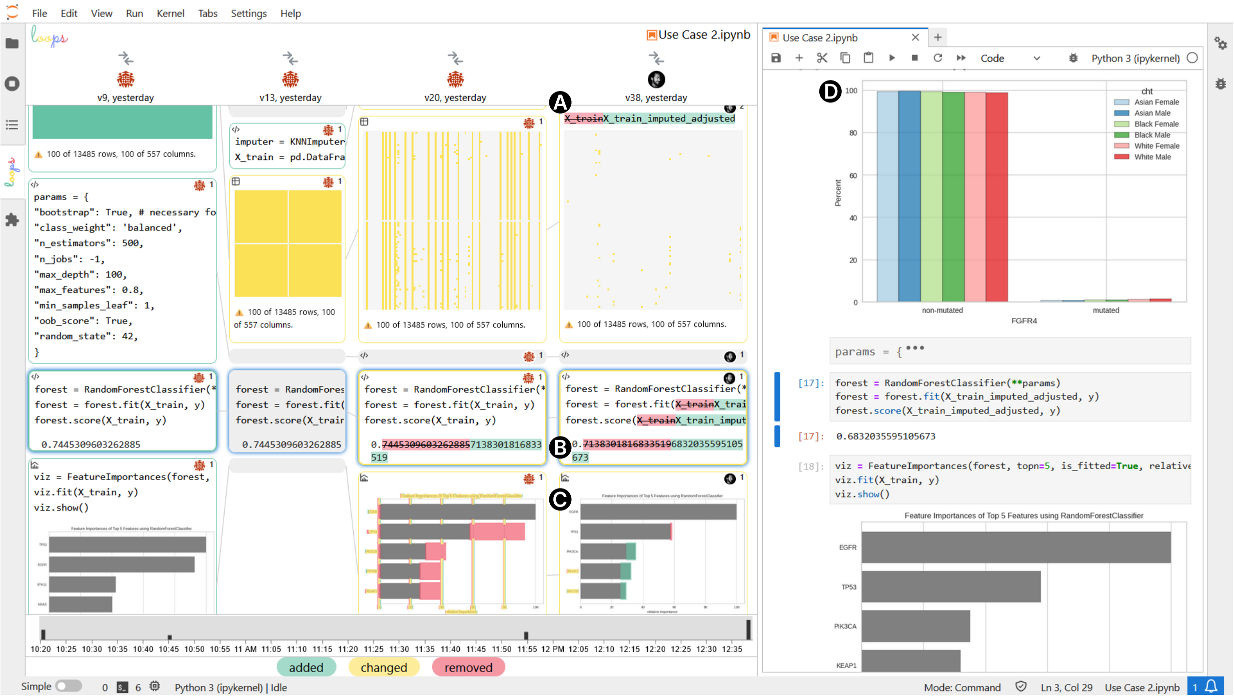
Pixel Similartiy: 97.6%

Regions Added: 13
Regions Removed: 0

(b) Visualization Difference

Fig. 16: Scatterplot of gross domestic product (GDP) per capita and life expectancy. In the updated version, labels are added for the three countries with the highest GDP per capita and the three countries with the highest life expectancy. The superimposed difference visualization shows how only the labels and its pointing arrows have been added, while the rest of the scatterplot remained unchanged.

B Full Page Screenshots of the JupyterLab Prototype



```
imputer = KNNImputer
X_train = pd.DataFra
```

```
params = {
"bootstrap": True, # necessary fo
"class_weight": 'balanced',
"n_estimators": 500,
"n_jobs": -1,
"max_depth": 100,
"max_features": 0.8,
"min_samples_leaf": 1,
"oob_score": True,
"random_state": 42,
}
```

~~X_train~~X_train_imputed_adjusted

```
forest = RandomForestClassifier(*
forest = forest.fit(X_train, y)
forest.score(X_train, y)

 0.7445309603262885
```

```
forest = RandomFores
forest = forest.fit(
forest.score(X_trair

 0.7445309603262885
```

```
forest = RandomForestClassifier(*
forest = forest.fit(X_train, y)
forest.score(X_train, y)
```

0.~~7445309603262885~~7138301816833519

forest = RandomForestClassifier(*
forest = forest.fit(~~X_train~~X_trai
forest.score(~~X_train~~X_train_input

0.~~7138301816833519~~6832035595105673

```
viz = FeatureImportances(forest,
viz.fit(X_train, y)
viz.show()
```

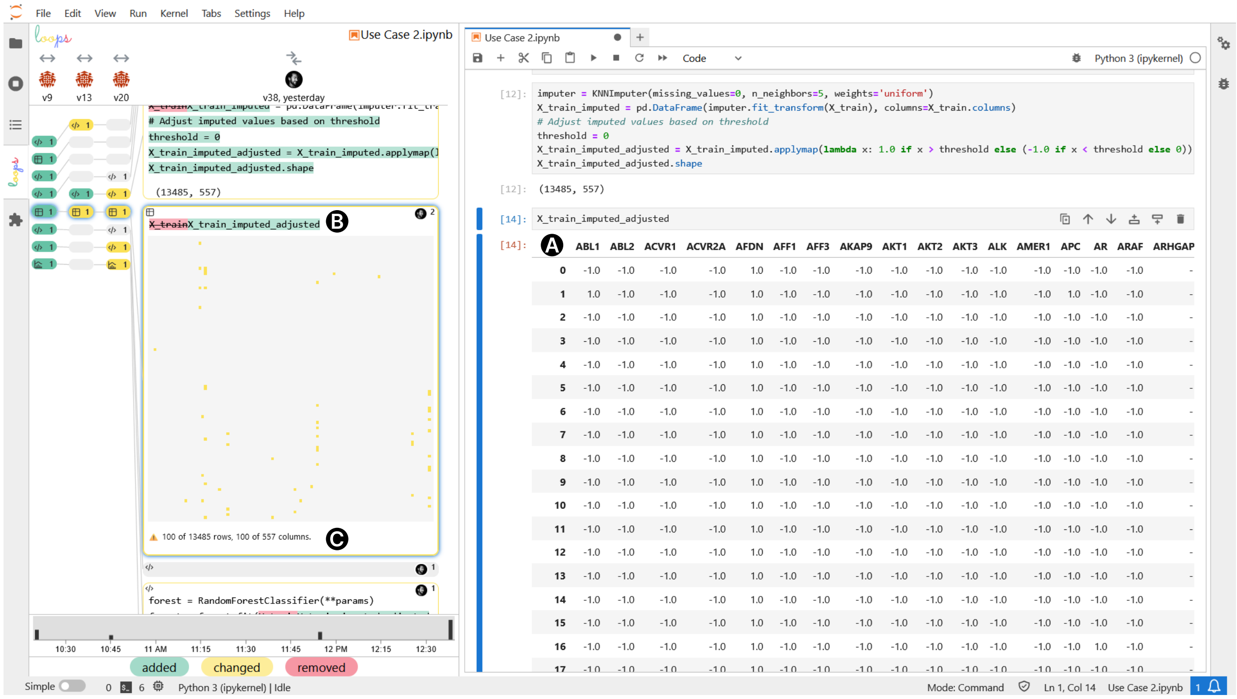
```
params = {•••
```

```
[17]: forest = RandomForestClassifier(**params)
      forest = forest.fit(X_train_imputed_adjusted, y)
      forest.score(X_train_imputed_adjusted, y)

[17]: 0.6832035595105673

[18]: viz = FeatureImportances(forest, topn=5, is_fitted=True, relative
      viz.fit(X_train, y)
      viz.show()
```

Fig. 17: Screenshot of JupyterLab with the Loops extension showing a part of the second use case (Sec. 5.2). The changes to the data substituion algorithm by analyst 👤 only had little impact on the data Ⓐ. Still, the models accuracy decreased Ⓑ slightly and its most important features changed. PIK3CA and KEAP1 are more important with this data and KMT2D is now part of the most important features instead of PTPRB. The histogram of FGFR4 Ⓓ shows that FGFR4 mutations are now almost equally distributed across the cohorts.



```
# Adjust imputed values based on threshold
threshold = 0
X_train_imputed_adjusted = X_train_imputed.applymap(l
X_train_imputed_adjusted.shape

 (13485, 557)
```

~~X_train~~X_train_imputed_adjusted

100 of 13485 rows, 100 of 557 columns.

```
forest = RandomForestClassifier(**params)
```

```
[12]: imputer = KNNImputer(missing_values=0, n_neighbors=5, weights='uniform')
      X_train_imputed = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.columns)
      # Adjust imputed values based on threshold
      threshold = 0
      X_train_imputed_adjusted = X_train_imputed.applymap(lambda x: 1.0 if x > threshold else (-1.0 if x < threshold else 0))
      X_train_imputed_adjusted.shape

[12]: (13485, 557)

[14]: X_train_imputed_adjusted
```

[14]:

	ABL1	ABL2	ACVR1	ACVR2A	AFDN	AFF1	AFF3	AKAP9	AKT1	AKT2	AKT3	ALK	AMER1	APC	AR	ARAF	ARHGAP
0	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-
1	1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-
2	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-
3	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-
4	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-
5	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-
6	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-
7	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-
8	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-
9	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-
10	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-
11	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-
12	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-
13	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-
14	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-
15	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-
16	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-
17	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-

Fig. 18: Screenshot of JupyterLab with the Loops extension showing a part of the second use case (Sec. 5.2). The large table shown in the notebook Ⓐ shows the imputed mutation data of the patient. The table is truncated to 100 of 557 columns and 100 of 13,485 rows. The tabular data difference visualization Ⓑ gives an overview over how the table's content changed through the applied imputation method. The warning below the difference visualization Ⓒ highlights that only a subset of the data was output in the notebook and that the visualization is thus incomplete.

C Interview Questions

C.1 Initial Questions

General questions on the usage of notebooks:

1. Do you use computational notebooks?
   (a) Which notebook environment are you using?
   (b) Which programming language are you using?
2. How often are you using notebooks?
3. Can you describe your typical workflow within a computational notebook? How do you organize your work, and what are the key steps in your process?

Questions about handling outputs in notebooks:

4. What do you output in notebooks?
5. How do you compare outputs?
6. When making changes to your notebook, how do you keep track of the modifications you've made, especially if you're working on complex analyses or data processing tasks?

Questions on how to deal with alternative approaches:

7. Do you try out alternatives in the notebook?
   (a) How do you compare these alternatives?
   (b) Do you delete the alternatives that turned out to be unsuccessful?
8. Can you share any examples of situations where you tried out alternative approaches or code in your notebook?
   (a) How did you evaluate their effectiveness, and what criteria did you use to decide whether to keep or discard them?
   (b) Were there situations where you would have needed removed code again at a later point?

Questions regarding collaboration with others:

9. Do you collaborate?
   (a) In real time?
10. When collaborating with others in your notebook work, how do you handle version control, conflict resolution, and sharing of results or insights?

Questions about version control and reproducibility:

11. Do you use version control? For what?
12. What challenges have you encountered related to reproducibility in your notebook-based work? How do you ensure that your work remains reproducible over time?
13. Can you describe any specific instances where you've faced difficulties in tracking the evolution of a notebook over time or understanding the impact of changes on your analyses?

C.2 Post-Demonstration Questions

1. After seeing a demonstration of the tool, can you share your initial impressions and thoughts on how it might be beneficial in your day-to-day work with computational notebooks?
2. Are there specific features or functionalities in the tool that you find particularly promising or valuable for your workflow? Could you elaborate on why these features stand out to you?
3. Can you envision any scenarios or tasks where this tool could help you save time, improve the reproducibility of your work, or enhance your ability to understand the history and changes in your notebooks?
4. Are there any specific challenges or obstacles you anticipate when trying to incorporate this tool into your current notebook workflow? What aspects of your workflow might be impacted?
5. Do you have any concerns about the learning curve or usability of the tool?
6. What other tools or extensions are you currently using to manage and visualize notebook changes, if any? How does this tool compare to those, and do you see potential advantages in switching or complementing your existing tools with this one?
7. Are there any specific use cases or scenarios in your work where you think this tool might not be as useful or relevant? What are the limitations or gaps you perceive?
8. If you were to suggest improvements or additional features for this tool based on your initial impression, what would you prioritize as the most valuable enhancements?
9. Is there something that should be highlighted/additionally visually indicated?
10. For visual outputs, would you prefer to see the previous version or a diff in the notebook history?
11. How do you see the focus on diffing only things that are visible in the notebook? I.e., if you show the head() of a dataframe you only see how the head changed (if at all), and miss that it might have changed somewhere else.