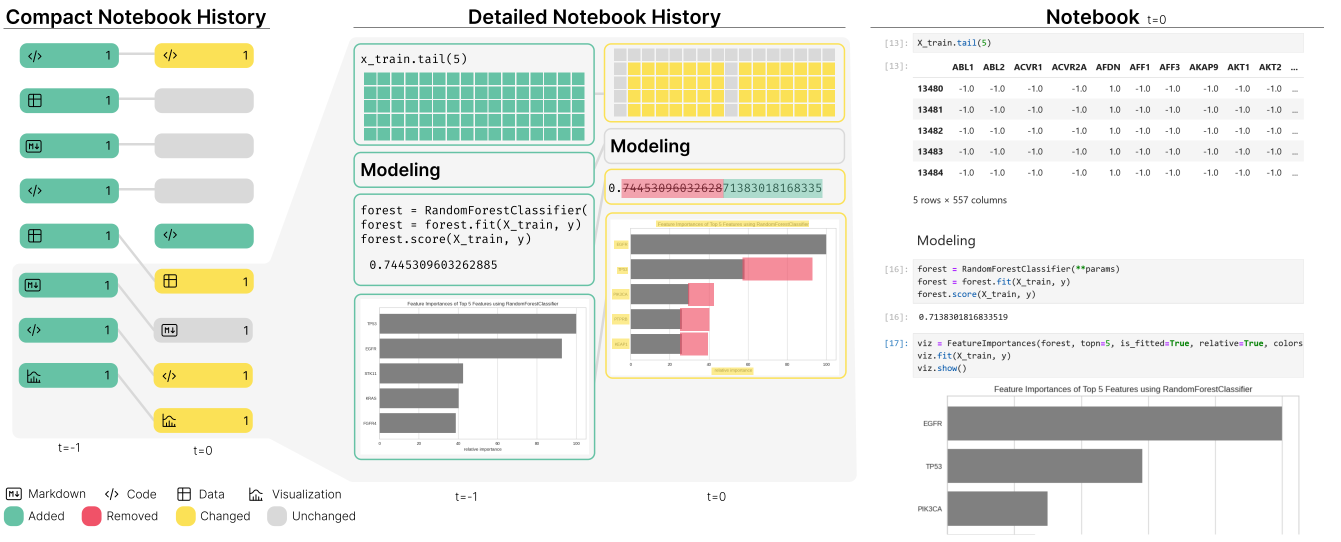
Eurographics Conference on Visualization (EuroVis) 2024
W. Aigner, D. Archambault, and R. Bujack
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 43 (2024), Number 3

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

Klaus Eckelt[1], Kiran Gadhave[2], Alexander Lex[2], and Marc Streit[1]

[1]Johannes Kepler University Linz, Austria
[2]University of Utah, United States



```
x_train.tail(5)
```

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

 0.7445309603262885
```

```
0.~~744530960326287~~1383018168335
```

```
[13]: X_train.tail(5)
```

[13]:

	ABL1	ABL2	ACVR1	ACVR2A	AFDN	AFF1	AFF3	AKAP9	AKT1	AKT2	...
13480	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	...
13481	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	...
13482	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	...
13483	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	...
13484	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	...

5 rows × 557 columns

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

[16]: 0.7138301816833519

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

Figure 1: *In this notebook, eight cells were created and executed ($t = -1$), before changing and re-executing cells, which updated the data and changed the model ($t = 0$). Loops visualizes the notebook history next to the notebook. The compact notebook history provides an overview on how the notebook's structure and content was changed over time. The rounded rectangles represent the notebook cells, color-coded according to their status: unchanged, changed, added, or deleted relative to the previous state. Inside executed cells, we also show the number of executions, and an icon indicating if the cell contains markdown, code, tables, or visualizations/images. The detailed notebook history also reveals how the cells' content changed, using difference visualizations specific to the various types of content present in notebooks.*

Abstract

Exploratory data science work is often described as an iterative process with cycles of obtaining, cleaning, profiling, analyzing, and interpreting data. These cycles create challenges within the linear structure of computational notebooks, leading to code quality, recall, and reproducibility issues. We present Loops, a set of visual support techniques for iterative and exploratory data analysis in computational notebooks. Loops leverages provenance information to provide direct feedback on the impact of changes made within the notebook. Through compact visual representations, we trace the evolution of the notebook over time, highlighting differences between versions. Detail views allow users to compare the cell content and output. Loops is compatible with various types of content present in notebooks, such as code, markdown, data, visualizations, or images. Loops not only improves the reproducibility of notebooks, but also supports analysts during their data science work by showing the effects resulting from changes and facilitating the comparison of multiple versions. We demonstrate our approach's utility and potential impact through two use cases and feedback from notebook users spanning various backgrounds.

CCS Concepts

• ***Mathematics of computing*** $\rightarrow$ *Exploratory data analysis;* • ***Human-centered computing*** $\rightarrow$ ***Information visualization;***

1. Introduction

Computational notebooks are the tool of choice in many data science applications [CFGT21]. As a literate programming tool [Knu84], computational notebooks allow analysts to combine code, output, and rich textual descriptions within a single document, combining narration of the process with execution. Notebooks are now capable of assembling incredibly diverse formats, including different programming languages (code), multimedia content as part of the narrative text, and interactive widgets and visualizations as part of the output. The most commonly used notebook technology is the Jupyter family [KRKP*16], the use of which has doubled annually in recent years [PZK*22].

The promise of literate programming—interspersing explanatory text with code—is reproducibility of the result. However, previous research has shown that reproducibility is by no means the norm when using computational notebooks [Guz20, PMBF19, WZC*19, WKLZ20]. A large-scale study by Pimentel et al. [PMBF19] 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" [Per21] (such as relying on out-of-order execution), become messy [HHB*19], and as a result, are often not reproducible [PMBF19, Guz20]. Data analysis is particularly affected by poor coding (and documentation) practices, as analysts often start with a vague understanding of their resources and goals [AZL*19, HWKP20]. 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 [PC05].

In the series of loops that make up the analysis process, each iteration refines the understanding and brings the analyst closer to their goal. This looping behavior makes it difficult for analysts to understand the state of their notebook. Analysts often update their code to reflect new insights or hypotheses, losing track of previous attempts and results [RTH18]. Alternatively, they may duplicate their code to compare different approaches, resulting in messy notebooks that contain outdated or redundant code [RTH18, WDBD21]. Furthermore, they may re-execute only parts of their notebooks, leading to out-of-order executions that break the logical flow and introduce hidden dependencies [RTH18, PMBF19, HHB*19]. These practices make it difficult to reproduce the analysis process and results for the analysts and others who want to reuse or verify their work.

Previous works tried to mitigate these issues by capturing the provenance of notebooks [SKR18, KJO*19] or providing live support for exploratory and/or iterative data science tasks [EJLW*22, EGMP23]. 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", highlights the changes made, and their impact on the output. We argue that 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 that content between different states 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. Data Science in Computational Notebooks

As an interdisciplinary field, data science uses scientific methods to extract insights from data [Sch21]. Figure 2 presents the data science workflow synthesized by Crisan et al. [CFGT21]. The non-linear process apparent from the figure is described as an "iterative conversation" [Per18] between the analyst and the data, wherein the analyst executes code, observes the outcomes, makes modifications, and repeats this cycle. Several studies describe the process as circular, non-linear, or opportunistic, often necessitating starting from scratch [Sch21, AZL*19, CFGT21].

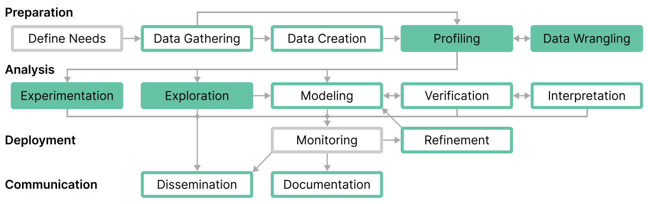


Figure 2: *The data science process, as presented by Crisan et al. [CFGT21]. Our approach has the most impact on the steps filled with green, while analysts also benefit from Loops in the steps with green outlines.*

Computational notebooks support such incremental and iterative analyses well, enabling analysts to edit, arrange, and execute small code blocks 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 over a third of the notebooks analyzed [PMBF19, Guz20]. In addition, the execution order cannot be precisely 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 [PMBF19]. These problems also occur when using conventional version control systems, and since the notebook contains outputs and meta-data, the differences are messy and confusing [CPH*20]. Also, when encountering dead ends, analysts often fail to document intermediate steps [RTH18, CPH*20]. Thus, notebooks are generally not suited to document their provenance.

Exploring the history of a notebook was rated the most challenging task after deployment in Chattopadhyay et al.'s survey [CPH*20]. When asked about remedies for tracking the evolution of a notebook, analysts favored automated versioning of their code and outputs [CPH*20]. However, tracking provenance alone is not sufficient. The tracked information must also be easily retrievable and easily digestible, for example, by relating it to a current state. 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 tested alternatives [CPH*20, AZL*19, RTH18]. These requirements have guided the design of our approach.

3. Related Work

We start this section by discussing research focused on tracking and visualizing the provenance of computational notebooks. While notebooks with tracking capabilities often support limited comparison of different versions of the same notebook and cells within the notebook versions, we dedicated a separate section to cover existing works on content-specific visual comparison aspects.

3.1. Tracking and Visualizing Notebook Provenance

Commercial platforms, such as Observable or Google Colab, offer cloud-based environments for analysts to work in notebooks. These platforms automatically track changes and store snapshots of notebooks on every change. JupyterLab creates a snapshot every time the user 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 datablocks or graphical output, including visualizations and images, are not supported. ProvBook [SKR18] is an extension to Jupyter Notebook that tracks the provenance of notebooks and allows users to compare individual cells to their previous versions, showing the previous code, the output it produces, and metadata. MLProvLab [KSKR21] tracks the notebook's provenance and displays each version as a graph, where the nodes are the notebook's cells and variables serve as the connecting edges. The graph's evolution over time can be inspected by browsing the executions. MARG [RSBB23] visualizes an annotated 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.

Most closely related to our work is Verdant [KJO*19], 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 of a revision 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. By clicking on a cell, users can inspect the history of that cell across all notebook revisions. In all views, only differences in the cell's input are highlighted.

In contrast to the works described above, our visualization of the notebook and its history resembles that of the user's notebook. We represent the notebook cells as rectangles aligned from top to bottom next to the user's notebook. Loops visualization of the notebook history shows the notebook structure, how it changes over time, if and how the cells have changed, and summarizes the difference between the various types of content in the notebook.

3.2. Visual Comparisons

The types of content present in notebooks are diverse. In this section, we discuss ways to compare them visually. We focus on static content in notebooks, i.e., plain and rich text, code, data, and images, excluding dynamic content such as audio, video and embedded websites.

Gleicher et al. [GAW*11] categorize comparative visualization into juxtaposition, superimposition, and explicit representation, applicable individually or in combination. Juxtaposition—presenting objects side-by-side or over time—can be easily implemented and applies to any visual representation but 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, also for rich text. Myers' algorithm [Mye86] is most commonly used to create human-readable text differences and is also the default option of git to show **code differences**. However, due to the unique structure of code, alternative approches have been shown to provide better results [NHM20]. 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, using a red/green color highlight, for example. The Git extension for JupyterLab [Jup23] also shows differences in code and plain text of juxtaposed notebooks. We support both views in Loops and use the existing work to visualize plain, rich, and code differences.

Gleicher [Gle18] 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, *feature* refers to the abstracted data (e.g., the height of a bar in a histogram), and *image* 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.

Current notebook environments and the related work discussed so far juxtapose graphical outputs in the notebook side-by-side without any explicit difference encoding. For superimposition, blending and color weaving techniques have been proposed to combine information from layered images [GAW*11, MHG10]. 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 [GW18, p. 87–90]. Pixel differences can be color-coded, for example, in red and green for negative and positive differences, respectively [HRTV06]. However, global translations, antialiasing, or compression in lossy formats can cause massive but irrelevant differences [HRTV06, Ima, Res23]. To suppress these artifacts, fuzzy difference methods can be applied that either take neighboring pixels into account [HRTV06], or allow a certain distance between the compared pixels in color space [Ima]. Resemble.js [Res23] offers a nuanced approach, calculating pixel-based image differences with options to ignore subtle or even all color differences, transparency, and/or antialiasing. It also provides different modes to visualize the amount of the difference. While the above methods allow for comparing two images, VAICO by Schmidt et al. [SGB13] supports more than two images. It identifies regions of differences (RoDs), applies hierarchical clustering to these RoDs, and creates an average of all compared images for contextual information. The RoDs are superimposed on the average image as colored polygons, which can be explored individually through interaction.

To identify differences in **data visualizations**, Ondov et al. [OJEF19] and Jardine et al. [JOEF20] evaluated the efficacy of superimposition and juxtaposition for various tasks in crowd-sourced experiments. While participants performed best with animated and superimposed images when it came to estimating correlations or identifying the largest difference between the images, juxtaposed images performed better in tasks where the image with the largest mean or range had to be identified [JOEF20]. The dependence on the user's task is also reflected in Diff in the Loop [WEDD22], where histograms can be compared in juxtaposed or superimposed views, or in a delta view that shows the distribution change. In Loops, two images or data visualizations can be juxtaposed or superimposed, and per-pixel differences can be explicitly highlighted. If the images/visualizations are superimposed, their opacity can be varied to alternate between them.

Vis-a-Vis [BB21] is an approach to visualizing 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 that shows the per-pixel variance of the outputs.

As data is fundamental to data science, **data differences** are at least equally important as code differences. Data sets evolve over time as analysts filter, clean, update, or expand it during their work. However, data difference visualizations are rarely integrated into the tools used by analysts. Sutton et al. [SHGC18] propose a method similar to text diffs that finds a human-readable transformation function from one data set to another, for example, `insert(i, C)` to insert column *C* at position *i*. In the following, however, we focus on the visualization of data differences. TACO [NSH*17] 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 [HWKP20] 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 changed. Eckelt et al. [EHA*23] visualize data sets that change over time as time curves [BSH*16]. 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 [WEDD22], 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 effects of these differences on the subsequent analysis are not visualized directly but through the difference visualizations in the following cells as soon as they are re-executed.

4. Loops Approach

In Loops, we leverage provenance information to visualize the notebook's history and the differences between notebook versions.

The notebook's history is visualized next to the notebook (see Figure 1). Changes to the notebook's content and structure are aggregated into states, which we display states either in (1) a compact representation that provides an overview of how the notebook has developed in terms of structure, time, and content; or (2) a detailed representation that reveals how the cells' content changed. We have created difference visualizations for the various types of content present in notebooks. These differences can be further explored in the comparison panel (see Figure 5).

The provenance is automatically recorded, as requested by analysts [CPH*20]. For this purpose, we store everything an analyst sees while working in the notebook and thus influence their information-foraging and sensemaking loops. The goal is not to collect comprehensive provenance that guarantees reproducibility—including details on the environment and hardware—but rather to support and make the analysis process comprehensible. Every time users execute a cell in the notebook and thus update it, we save the cells of the notebook, including their order, which cell was active and executed, their input and output, and how these inputs and outputs were displayed.

4.1. Visualizing and Comparing Notebook States

The recorded provenance is a list of changes and executions that can quickly become uninterpretable and overwhelming [CPH*20]. Therefore, we have considered how the provenance states can be best aggregated. Previous work has grouped notebook states in temporal proximity [KM18]. For Loops, however, we decided to take a different approach and by default group all states that are executed in linear order. This means that states are 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 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 [RSBB23]. Going back in the notebook—and re-evaluating already executed parts — corresponds to a new iteration of the analysis loop. In Figure 1, a new state is introduced as the analyst edits

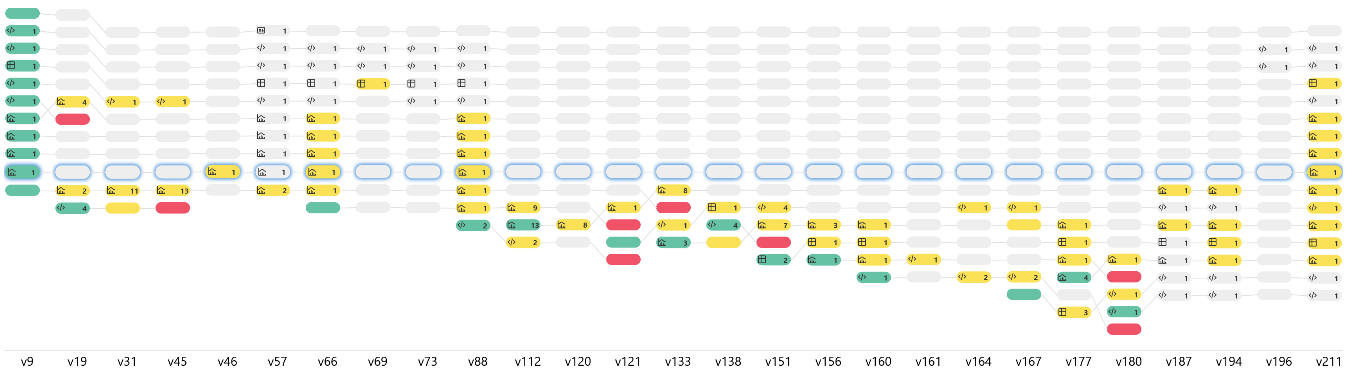


Figure 3: *Compact presentation of an analysis over several weeks. A rounded rectangle represents each notebook cell, color-coded ● green for insertions, ● red for removals, ● yellow for content changes, and ● gray for unchanged cell content. Inside executed cells, we show how often each cell was executed plus an icon indicating if the cell contains markdown or code. For code cells, we additionally adapt the icon if the cell outputs a table or visualization/image. In the first state, represented as the first column of the history, the analyst created and executed multiple cells. The cells at the end of the notebook were edited iteratively, and the entire notebook was executed again only a few times. The cells of the states are aligned around the active cell, emphasized with a blue shadow.*

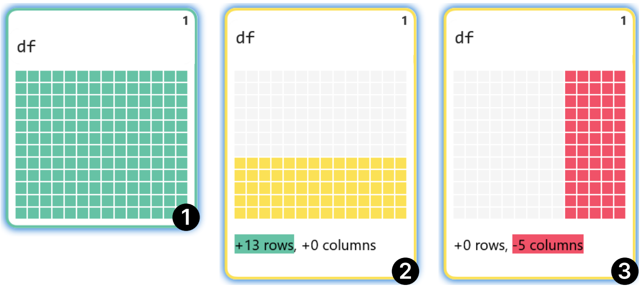


Figure 4: *History of a code cell that outputs data. The cell was created in the first state ❶, recognizable by the green border. In the second state ❷, 13 rows were added, of which five are shown in the notebook. In the third state, five columns were removed ❸.*

and re-executes the first cell of the notebook again, before updating the data processing step and re-running the modeling step.

The aggregated states are visualized and arranged horizontally according to their creation order. Each cell is represented as a rectangular block with rounded corners, akin to a notebook cell. The cells are arranged vertically, identical to the notebook. Cells of adjacent states are connected with lines. Each state offers two levels of detail: a compact and a detailed representation. The compact representation provides an overview of the state, indicating added, changed, executed, and deleted cells. The detailed representation also reveals the cells' content and how they have changed compared to the previous state. By visualizing the structure of the notebook and how it changed, we address the analysts' need for a retrievable and comparable history [CPH*20].

4.1.1. Compact State Representation

In the compact state representation, shown in Figure 3, gives an overview of the evolution of the notebook by showing all cells in the same small size. We embed icons that indicate whether a cell contains Markdown or code. For 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 the aggregated state. This helps identify the parts of the notebook that have been modified the most frequently or not executed at all. Cells are color-coded according to their status: unchanged, changed, added, or deleted relative to the previous state. 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 (compare Figure 3).

The active cell also serves as the anchor point for aligning the other cells. When the analyst changes the active cell in the notebook, our provenance visualization re-aligns the cell's history vertically for better comparison.

Due to the small width, we can display many states on the screen, depending on the resolution. Figure 3 shows an analysis carried out over several weeks with 211 versions and 27 states. Due to the small cell height, we can display even larger notebooks in their entirety without scrolling [RTH18].

4.1.2. Detailed State Representation

The detailed representation, shown in Figure 1, visualizes the changes in cell input and output that occurred in a state, allowing analysts to better understand their changes' impact. By default, Loops uses the compact representation for all but the latest aggregated state, but users can freely switch between the representations.

In the detailed state representation, we utilize a degree-of-interest (DoI) function [Fur06] to determine the content that can be displayed for each cell. This function is applied to both the input and the output (where applicable). Markdown cells with headlines and code cells that output visualizations or images are assigned a DoI of 1, which indicates high interest. These elements provide structure to the document, and their full display in the detailed state representation facilitates navigation [RTH18, CPH*20]. Similarly, the currently active cell is also assigned a DoI of 1, allowing easy tracking of the cell's history, as illustrated in Figure 4. For all other inputs and outputs, the DoI is set to 1 if there was a change, and zero otherwise. The content of the inputs and outputs of cells with a DoI of 0 is not displayed. Cells where both the input and the output have a DoI of 0 are shown in the compact representation. This approach allows the state representation to focus on changes and their impact across the notebook. When the DoI is 1, a difference visualization for the respective content type is displayed, which color-codes changes, and the colored background indicating the cell's state is replaced with a colored frame. Individual difference visualizations provide further details and are described in the following section. Due to space constraints in the state representation, the difference visualizations are reduced in detail: Data diffs do not show the actual content but indicate additions, removals, and changes; and for image and visualization diffs, the size of highlighted change regions is increased (at the expense of unchanged regions) to ensure visibility in the thumbnail-like display.

4.2. Comparison of Cells and Their Output

Due to the limited size, the difference visualizations used in the detailed state representation (Section 4.1.2) can only hint at changes. To access a more comprehensive view, users can open a separate comparison panel from the history of each cell. This panel features large difference visualizations that display the full content of the cell's input and outputs. When the comparison panel is opened, the compared cell is set as the active cell. This aligns the notebook and the history for easy reference.

The panel allows for switching between the cell input and any existing output. The selection of the appropriate difference visualization is automatic and based on the content type. A code-comparison view is displayed for cell inputs. For outputs, the difference visualization varies based on the type: plain text, rich text (from Markdown, for example), code, tables, images, or data visualizations. All difference visualizations can display the two compared versions side-by-side (juxtaposed) or unified (superimposed) in a single view, each with explicit encoding of differences, offering flexibility in viewing and comparing changes. In the side-by-side comparison, the two versions are displayed separately. Deletions are highlighted in one version, and insertions are highlighted in the other. In the unified comparison, a single visualization incorporates both versions and their differences. In both, changes are explicitly encoded by applying the same color-coding that is used for the cells: ● green and ● red for insertions and removals, and ● yellow for content changes (that is, of table cells or pixels of the image).

4.2.1. Text and Code Comparison

For the visualization of text and code differences, we based our designs on the standard methods used in word processors and editors.

Text comparison can be performed at different levels: character, word, or line level. Our difference visualization uses the Myers difference algorithm [Mye86], a widely used algorithm 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 and easier 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 Figure 5). These additional steps account for the unique characteristics that the code presents in contrast to text documents to ensure that the differences are meaningful and relevant. In the detailed state representation, syntax highlighting is disabled so that only the changes are emphasized by color (see Figure 1).

4.2.2. Data Comparison

Understanding how data changes over the course of an analysis is crucial. It may be through iterations by the analyst or through external updates to the data [AZL*19, MLW*19, HWKP20].

In Loops, these changes are explicitly visualized. Inspired by TACO [NSH*17], our visualization compares two data tables as they are output in the notebook, identifying changes in the table cells and adding and removing columns or rows. Columns or rows that are added or removed are highlighted in green and red, respectively, while table cells with changed content are shown in yellow (see Figure 4). Changing the order of columns is not considered a change, as the columns' data remains unchanged.

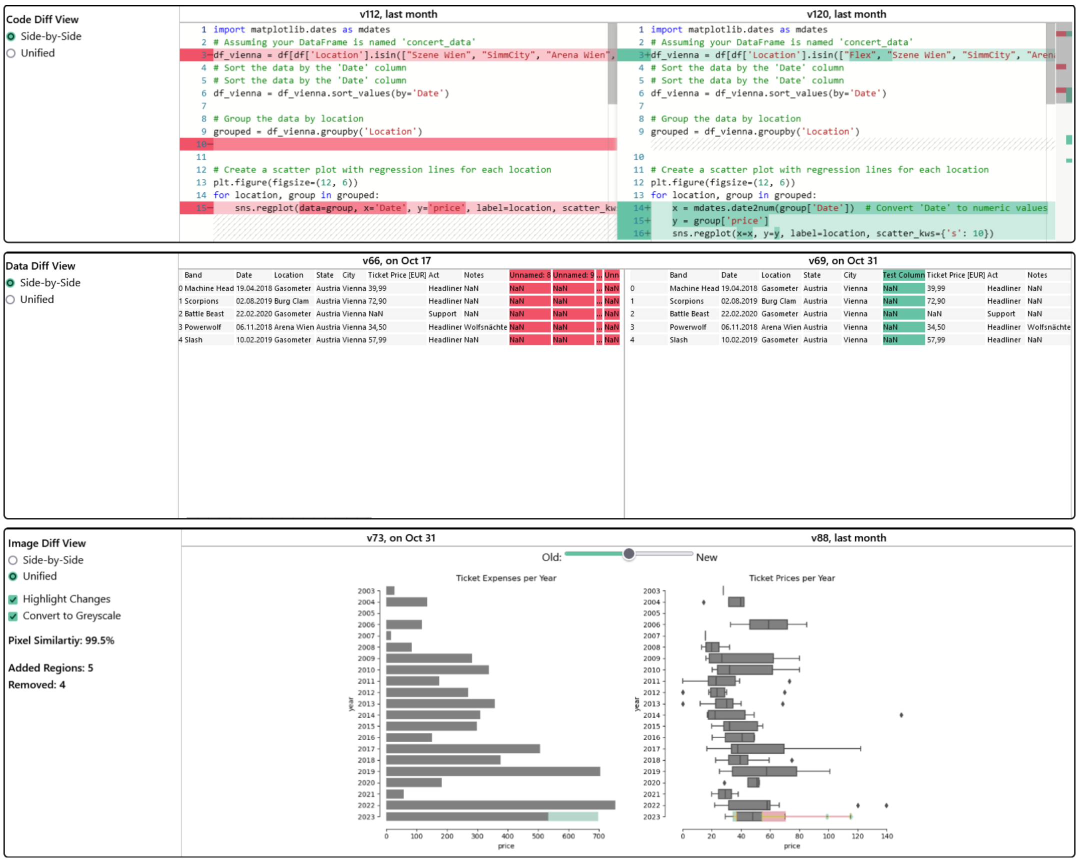
For the state representation, the raw data is not shown. Instead, we represent all table cells in uniform size, color-coded by the type of change (see Figure 4). The raw data of the table cells is shown in the comparison panel (see Figure 5). We render removed data with a strike-through font styling without further color-coding. Compared to regular text, we use a different visualization approach to avoid nesting the colors representing the change type.

4.2.3. Comparison of Images and Visualizations

In Loops, we compare them at the image data 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 data level, the image difference visualization is part of a broader context that includes differences of all other steps leading to the image change. This could be changes in the data used to create a visualization or alterations in the code that defines and styles the visualization.

Balancing the needs for comparison between generic images (photos, diagrams) and data visualizations is difficult. Given the data science focus of this work, we prioritize the requirements for comparing data visualization over those for generic images. For instance, a slight change in brightness may not matter in a photograph but can be crucial in a heatmap. Image subtraction is used routinely for enhancing differences between images [GW18, p. 87] and is useful when changes in successive images need to be detected [BB08, 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 images are first padded to the same size if their sizes vary, with the added pixels initialized as white and opaque. The padded pixels are inserted at the top and right borders. Data visualizations typically have their axes at the left and bottom borders, so padding at the top and right ensures that the axes



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

v66, on Oct 17

	Band	Date	Location	State	City	Ticket Price [EUR]	Act	Notes	Unnamed: 8	Unnamed: 9	...	Unn
0	Machine Head	19.04.2018	Gasometer	Austria	Vienna	39,99	Headliner	NaN	NaN	NaN	...	NaN
1	Scorpions	02.08.2019	Burg Clam	Austria	Vienna	72,90	Headliner	NaN	NaN	NaN	...	NaN
2	Battle Beast	22.02.2020	Gasometer	Austria	Vienna	NaN	Support	NaN	NaN	NaN	...	NaN
3	Powerwolf	06.11.2018	Arena Wien	Austria	Vienna	34,50	Headliner	Wolfsnächte	NaN	NaN	...	NaN
4	Slash	10.02.2019	Gasometer	Austria	Vienna	57,99	Headliner	NaN	NaN	NaN	...	NaN

v69, on Oct 31

	Band	Date	Location	State	City	Test Column	Ticket Price [EUR]	Act	Notes
0	Machine Head	19.04.2018	Gasometer	Austria	Vienna	NaN	39,99	Headliner	NaN
1	Scorpions	02.08.2019	Burg Clam	Austria	Vienna	NaN	72,90	Headliner	NaN
2	Battle Beast	22.02.2020	Gasometer	Austria	Vienna	NaN	NaN	Support	NaN
3	Powerwolf	06.11.2018	Arena Wien	Austria	Vienna	NaN	34,50	Headliner	Wolfsnächte
4	Slash	10.02.2019	Gasometer	Austria	Vienna	NaN	57,99	Headliner	NaN

Image Diff View
Side-by-Side [✓ Unified]

✓ Highlight Changes
✓ Convert to Greyscale

Pixel Similartiy: 99.5%

Added Regions: 5
Removed: 4

v73, on Oct 31

v88, last month

Figure 5: *Detailed code, data, and image/data visualization differences. The data difference visualization shows that multiple "Unnamed" columns were removed from the data, while a "Test Column" is added. The difference of the data visualization shows changes in the two concatenated charts: While the bar chart's last bar became longer, the last quartile range of the last box plot becomes smaller.*

remain aligned. The images are then subtracted and subsequently turned into grayscale by calculating the perceived luminance of each pixel [BB08, p. 256]. 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 state representation 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. The calculated difference is then transformed into an RGB image. Pixels that changed from white to a different color are highlighted in green, from color to white in red, and from one color to another in yellow (see Figure 5). We also calculate the bounding boxes of the changes. In initial experiments, we used rectangular or convex hull bounding boxes to highlight changed areas, which made the changes within the bounding box 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.

Users can choose whether the images with explicit difference encoding are displayed side-by-side or in unified mode. In the unified difference visualization, the images are combined with alpha blending, and the opacity levels can be controlled with a slider (see Figure 4).

We found that whether unified or side-by-side layouts are better is highly task-dependent. Generally, superimposition used by our unified difference visualizations seems most helpful if the compared data is similar enough [JOEF20, GAW*11]. To determine similarity and decide on the layout to use by default, we evaluated multiple image similarity metrics: (*i*) Structured Similarity Index (SSIM) [WBSS04], (*ii*) Normalized Mututal Information (NMI) [SHH99], and (*iii*) Oriented FAST and Rotated BRIEF (ORB) [Bra00]. In our experiments, however, these metrics were not suited for application to data visualizations due to their different purpose of application (SSIM, NMI) and the lack of unique features in marks (ORB). Instead, we divide the number of changed pixels by the total number of pixels. The resulting relative pixel similarity is also displayed next to the difference visualization. We present the difference visualizations in a unified layout as long as the relative similarity of the pixels is at least 90%. 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 aids. In the detailed comparison, both can be changed by the user at any time, as can the conversion of the images to grayscale so that only the explicit difference encoding is colored. In the state representation, we always show the unified difference visualization with the images in grayscale (see Figure 1).

5. Implementation

We have implemented the open source prototype of Loops as an extension for JupyterLab that is available through JupyterLab's built-in extension manager and 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 notebooks with interesting provenance information. Although the screenshots and use cases presented in this work use Python, Loops 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.

The extension adds our notebook overview visualization to JupyterLab's side panel and waits for user interactions with the notebook, after which the new state of the notebook is recorded. We use Trrack [CGL20] to store the notebook states as a provenance graph. When the user saves the notebook, the provenance graph is embedded in the notebook's metadata, such that it does not have to be transferred separately. For the difference visualizations, we use html-diff [Tan23] for rich text differences, the Monaco Editor [Mic23] for code differences, D3 [BOH11] to create the data difference visualizations, and OpenCV [Bra00] to calculate and visualize image differences.

6. 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 Section 7.1.

6.1. 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 (see Figure 3). The data set is a long-form table with 501 entries from 2003 to 2023, with features describing the artist, date, location, ticket price, and act (headliner/support). The history contains 211 versions of the notebook, represented in 27 states. The final notebook consists of 16 cells, which is a typical size for notebooks [RTH18].

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. We started our analysis on October 6th by importing the concert data and applying a few initial data wrangling steps. We 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 we 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. After updating the data, the visualization cell was changed up to 13 times in one state. As changes are aggregated as long as they are made from top to bottom, our state representation shows the visualization in which a changed data representation resulted.

We re-executed the notebook on October 17th and 31st. The detailed state representations reveal 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 track changes by clearly highlighting output differences.

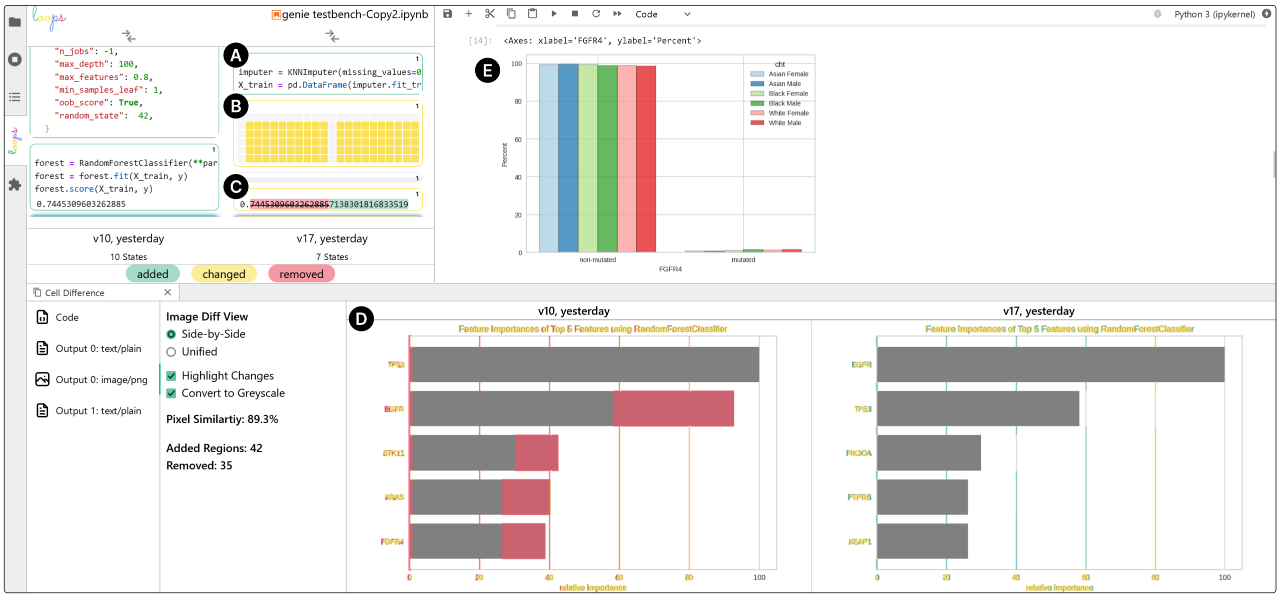
The most extensive phase of our analysis occurred on November 6th, involving 123 of 201 versions. We extended the visualization of price trends over time with regression lines and adjusted prices for inflation by including a data set containing inflation rates.This required several iterations, and we used Loops to verify the changes using the difference visualizations shown in Figure 5.

In the last change on November 27th, we executed the entire notebook, which caused the visualizations to change due to the updated concert and inflation data.

The compact history of the notebook shown in Figure 3 provides several insights: (1) This first analysis step is represented in the first five states of Figure 3 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 states 57, 66, 88, and 211 of Figure 3 that correspond to October 16th, 17th, November 6th, and 27th, respectively. (3) The history also shows that during the extensive analysis on November 6th, no changes were made to previously created cells.

6.2. 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 [EAB*22,AAC17]. We aimed to validate the mutational differences between these cohorts, as reported in the literature [NAK21]. 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



```
    "n_jobs": -1,
    "max_depth": 100,
    "max_features": 0.8,
    "min_samples_leaf": 1,
    "oob_score": True,
    "random_state":  42,
  }
```

```
forest = RandomForestClassifier(**par
forest = forest.fit(X_train, y)
forest.score(X_train, y)
0.7445309603262885
```

```
imputer = KNNImputer(missing_values=0
X_train = pd.DataFrame(imputer.fit_tr
```

```
0.~~7445309603262885~~71383018168335 19
```

```
[14]: <Axes: xlabel='FGFR4', ylabel='Percent'>
```

v10, yesterday
10 States

v17, yesterday
7 States

Output 0: text/plain
Output 0: image/png
Output 1: text/plain

Image Diff View
[✓ Side-by-Side]
Unified

[✓ Highlight Changes]
[✓ Convert to Greyscale]

Pixel Similartiy: 89.3%

Added Regions: 42
Removed: 35

Figure 6: *Analysis of cancer patient data in JupyterLab with Loops. The visualization of the notebook's evolution shows two states corresponding to the initial execution of the notebook and the following what-if analysis. The second state adds a cell to substitute missing values* Ⓐ*, which changed the data* Ⓑ*, the model's accuracy* Ⓒ*, and the features that are most important to distinguishing cohorts* Ⓓ*. In the notebook, the histogram shows that FGFR4 mutations are now almost equally distributed across the cohorts* Ⓔ*.*

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 [Kos23].

We begin our analysis by replicating our previous findings. We import the necessary libraries, load the cancer patient cohort data, and inspect the data set. As our goal is to identify differences between cohorts using a random forest model, we separate the features used for the prediction and the cohort labels into distinct variables. We 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, we again inspect the data set to verify the changes. We also create a visualization to show the distribution of FGFR4 genetic mutations among the analyzed cohorts. After preparing the data, we set hyperparameters, train the random forest model, and output its accuracy. We create a bar chart to visualize the model's most important features, and the results align with our previous analysis.

Next, we substitute the missing data and assess how the results of the analysis change. We extend our initial feature engineering step and substitute the missing genetic data with a nearest-neighbor approach using the patient's data with the most similar genetic profile (Figure 6 Ⓐ). As we re-run the subsequent steps in our notebook, the changes in the data are reflected in the output (Figure 6 Ⓑ) and the histogram of the FGFR4 mutations, which are now almost equally distributed across the cohorts (Figure 6 Ⓔ). A slight decrease in the model's accuracy can be observed in the state difference, suggesting that previously missing data played a role in the differentiation of the cohorts (Figure 6 Ⓒ). In the new model, FGFR4 is no longer one of the most important feature distinguishing cohorts. 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 (Figure 6 Ⓓ).

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 future repetition. With loops, any resulting output differences can be quickly identified.

7. Discussion

In this section, we first summarize the feedback we have received from notebook users, and then go on to discuss the challenges encountered and avenues for potential improvements.

7.1. Qualitative User Feedback

We gathered feedback on our approach by interviewing notebook users with various backgrounds: a student (S) of AI, a professional data scientist (DS), a researcher (R) working in the field of human-centered AI, and a lecturer (L) teaching undergraduate and graduate students in the field of AI.

In the interviews, we first asked them questions about how they work in notebooks. A common thread among S, DS, and R is the creation of multiple cells, each containing code with slight variations to facilitate the comparison of alternatives. When asked about how they deal with alternatives once they had picked a solution, none of the participants could articulate specific criteria they apply. S and DS leaned towards retaining all code, even if it resulted in cluttered notebooks. Notably, all four participants had previously experienced that they needed code they had deleted. The four notebook users mainly work alone on notebooks; collaboration only takes place asynchronously, if at all. These responses are consistent with the results of previous studies [RTH18, AZL*19, CPH*20].

As a next step in the interview, we showed them a notebook we had prepared based on the concert data analysis. We explained the data, the analysis goals, and the notebook structure. We then opened the compact representation of the analysis history. Participants stated that the color-coding with green for new and red for removed cells was obvious, yet, they were not sure 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. We also asked them to identify the cells that changed the most. After exploring the compact state representation, 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 output. Since participants were not yet familiar with the data, they could not tell how outputs changed. We changed the last state to the detailed representation, revealing the changes in data and visualizations. We also opened the comparison panel for multiple cells to demonstrate the text, code, data, and image difference visualizations. The participants recognized how the familiar encoding of code differences was translated to other types of content. They consider the difference visualizations to be most useful when the changes are subtle. Participant R creates large composite figures and finds the difference visualization to be particularly useful when styling these figures to identify elements that have not been updated.

Overall, participants found the visualization of state and cell content differences intuitive. However, for data, DS, R, and L expressed a desire to extend the exploration of differences beyond the visible subset to the entire data. Although Loops currently does not show these, we indicate changes in the size of the data in addition to the changes in the visible content (see Figure 4). R and S suggested an onboarding process for features and, most importantly, interactions. L considered the integration of the analysis process into the grading of assignments. 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 contributors to changes.

7.2. Limitations

We have demonstrated the utility of our approach in the use cases and received positive feedback from potential users. In the following, we discuss scalability limitations of our technique regarding long version histories, notebooks with a large number of cells, and notebook cells with extensive content.

Scalability. The compact state representation scales well with the increasing number of cells. However, the number of states depends on the user's execution patterns, and horizontal space is limited. To enhance temporal understanding, we plan to incorporate a continuous time scale alongside the discrete time steps of the states. This addition will allow users to identify periods during which the analysis was carried out or paused. Visualizing code differences within the state representation also runs up against spatial constraints, often requiring users to scroll to examine variations. Our attempts to reduce code differences to the neighborhood of changed content encountered challenges, especially when exploring a cell's history. To address this, we are considering a more nuanced degree-of-interest function to control the amount of visualized code in a changed state and its adjacent states [AHSS14]. Our pixel-based approach to visualize differences between images is also constrained, particularly with images of different sizes and shifts within the image. Figure 6 shows that the title, grid lines, and x-axis change as they shift slightly to the left. Scaling up visualizations results in many differences, even when the content remains unchanged. We argue that data visualizations could benefit from a specialized difference approach that is aware of the visualized data, to ensure that the difference visualizations are robust to style changes.

Collaboration and Reactive Notebooks. Collaboration and co-authoring of notebooks is an important topic in data analysis [Kos23]. The notebook users we interviewed also share their notebooks with others and usually take turns working on the notebook with their colleagues. Some scholars argue that reactivity is a prerequisite for real-time collaboration [Per21]. 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, changing a single cell can result in changing many outputs. Loops can help users recognize and understand these changes. Second, tracking the provenance is still important to understand the analysis evolution or to compare alternatives. To visualize who contributed to changes in the notebook, we consider expanding the state representations with avatars of the collaborators. For real-time collaboration, it is necessary to adapt the aggregation of notebook versions when users work on different notebook parts.

8. Conclusion

Analysts face challenges and pain points when working with notebooks [CPH*20]. 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 (see Figure 2). Based on the feedback from four notebook users, we are confident that Loops can support data science processes in notebooks effectively.

References

[AAC17] AACR Project Genie Consortium: AACR Project GENIE: powering precision medicine through an international consortium. *Cancer discovery 7*, 8 (2017), 818–831. doi:10/ggnnrn. 8

[AHSS14] Abello J., Hadlak S., Schumann H., Schulz H.-J.: A Modular Degree-of-Interest Specification for the Visual Analysis of Large Dynamic Networks. *IEEE Transactions on Visualization and Computer Graphics 20*, 3 (2014), 337–350. doi:10.1109/TVCG.2013.109. 10

[AZL*19] Alspaugh S., Zokaei N., Liu A., Jin C., Hearst M. A.: Futzing and Moseying: Interviews with Professional Data Analysts on Exploration Practices. *IEEE Transactions on Visualization and Computer Graphics 25*, 1 (2019), 22–31. doi:10.1109/TVCG.2018.2865040. 2, 3, 6, 10

[BB08] Burger W., Burge M. J.: *Digital Image Processing: An Algorithmic Introduction Using Java*, 1 ed. Texts in computer science. Springer, 2008. 6, 7

[BB21] Bolte F., Bruckner S.: Vis-a-Vis: Visual Exploration of Visualization Source Code Evolution. *IEEE Transactions on Visualization and Computer Graphics 27*, 7 (2021), 3153–3167. doi:10.1109/TVCG.2019.2963651. 4

[BOH11] Bostock M., Ogievetsky V., Heer J.: D3: Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 2301–2309. doi:10.1109/TVCG.2011.185. 8

[Bra00] Bradski G.: The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000). 7, 8

[BSH*16] Bach B., Shi C., Heulot N., Madhyastha T., Grabowski T., Dragicevic P.: Time Curves: Folding Time to Visualize Patterns of Temporal Evolution in Data. *IEEE Transactions on Visualization and Computer Graphics 22*, 1 (2016), 559–568. doi:10.1109/TVCG.2015.2467851. 4

[CFGT21] Crisan A., Fiore-Gartland B., Tory M.: Passing the Data Baton : A Retrospective Analysis on Data Science Work and Workers. *IEEE Transactions on Visualization and Computer Graphics 27*, 2 (2021), 1860–1870. doi:10.1109/TVCG.2020.3030340. 2

[CGL20] Cutler Z. T., Gadhave K., Lex A.: Trrack: A Library for Provenance Tracking in Web-Based Visualizations. In *IEEE Visualization Conference (VIS)* (2020), pp. 116–120. doi:10.1109/VIS47514.2020.00030. 8

[CPH*20] Chattopadhyay S., Prasad I., Henley A. Z., Sarma A., Barik T.: 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* (New York, USA, 2020), CHI '20, Association for Computing Machinery, pp. 1–12. doi:10.1145/3313831.3376729. 2, 3, 4, 5, 10

[EAB*22] Eckelt K., Adelberger P., Bauer M. J., Zichner T., Streit M.: 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. 8

[EGMP23] Epperson W., Gorantla V., Moritz D., Perer A.: Dead or Alive: Continuous Data Profiling for Interactive Data Science. *IEEE Transactions on Visualization and Computer Graphics* (2023), 1–11. doi:10.1109/TVCG.2023.3327367. 2

[EHA*23] Eckelt K., Hinterreiter A., Adelberger P., Walchshofer C., Dhanoa V., Humer C., Heckmann M., Steinparz C., Streit M.: Visual Exploration of Relationships and Structure in Low-Dimensional Embeddings. *IEEE Transactions on Visualization and Computer Graphics 29*, 7 (2023), 3312–3326. doi:10.1109/TVCG.2022.3156760. 4

[EJLW*22] Epperson W., Jung-Lin Lee D., Wang L., Agarwal K., Parameswaran A. G., Moritz D., Perer A.: Leveraging Analysis History for Improved In Situ Visualization Recommendation. *Computer Graphics Forum 41*, 3 (2022), 145–155. doi:10.1111/cgf.14529. 2

[Fur06] Furnas G. W.: A Fisheye Follow-up: Further Reflections on Focus + Context. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2006), CHI '06, ACM, pp. 999–1008. doi:10.1145/1124772.1124921. 5

[GAW*11] Gleicher M., Albers D., Walker R., Jusufi I., Hansen C. D., Roberts J. C.: Visual comparison for information visualization. *Information Visualization 10*, 4 (2011), 289 –309. doi:10.1177/1473871611416549. 3, 7

[Gle18] Gleicher M.: Considerations for Visualizing Comparison. *IEEE Transactions on Visualization and Computer Graphics 24*, 1 (2018), 413–423. doi:10.1109/TVCG.2017.2744199. 3

[Guz20] Guzharina A.: 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: 2023-11-30. 2

[GW18] Gonzalez R., Woods R.: *Digital Image Processing, Global Edition*. Pearson Education, 2018. 3, 6

[HHB*19] Head A., Hohman F., Barik T., Drucker S. M., DeLine R.: Managing Messes in Computational Notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (New York, USA, 2019), CHI '19, Association for Computing Machinery, pp. 1–12. doi:10.1145/3290605.3300500. 2

[HRTV06] Hollingsworth B. V., Reichenbach S. E., Tao Q., Visvanathan A.: Comparative visualization for comprehensive two-dimensional gas chromatography. *Journal of Chromatography A 1105*, 1 (2006), 51–58. doi:10.1016/j.chroma.2005.11.074. 3

[HWKP20] Hohman F., Wongsuphasawat K., Kery M. B., Patel K.: Understanding and Visualizing Data Iteration in Machine Learning. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (New York, USA, 2020), CHI '20, Association for Computing Machinery, pp. 1–13. doi:10.1145/3313831.3376177. 2, 4, 6

[Ima] ImageMagick Studio LLC: Comparing – ImageMagick Examples. https://imagemagick.org/Usage/compare/. Accessed: 2023-11-30. 3, 4

[JOEF20] Jardine N., Ondov B. D., Elmqvist N., Franconeri S.: The Perceptual Proxies of Visual Comparison. *IEEE Transactions on Visualization and Computer Graphics 26*, 1 (2020), 1012–1021. doi:10.1109/TVCG.2019.2934786. 4, 7

[Jup23] Jupyter Development Team: jupyterlab-git. https://github.com/jupyterlab/jupyterlab-git, 2023. Accessed: 2023-11-30. 3

[KJO*19] Kery M. B., John B. E., O'Flaherty P., Horvath A., Myers B. A.: Towards Effective Foraging by Data Scientists to Find Past Analysis Choices. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (New York, USA, 2019), CHI '19, Association for Computing Machinery, pp. 1–13. doi:10.1145/3290605.3300322. 2, 3

[KM18] Kery M. B., Myers B. A.: Interactions for Untangling Messy History in a Computational Notebook. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2018), pp. 147–155. doi:10.1109/VLHCC.2018.8506576. 4

[Knu84] Knuth D. E.: Literate Programming. *The Computer Journal 27*, 2 (1984), 97–111. doi:10.1093/comjnl/27.2.97. 2

[Kos23] Kosara R.: Notebooks for Data Analysis and Visualization: Moving Beyond the Data. *IEEE Computer Graphics and Applications 43*, 1 (2023), 91–96. doi:10.1109/MCG.2022.3222024. 9, 10

[KRKP*16] Kluyver T., Ragan-Kelley B., Pérez F., Granger B., Bussonnier M., Frederic J., Kelley K., Hamrick J., Grout J., Corlay S., Ivanov P., Avila D., Abdalla S., Willing C., Jupyter Development Team: Jupyter Notebooks—a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. IOS Press, 2016, pp. 87–90. doi:10.3233/978-1-61499-649-1-87. 2

[KSKR21] Kerzel D., Samuel S., König-Ries B.: Towards Tracking Provenance from Machine Learning Notebooks. In *Proceedings of the 13th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - KDIR,* (2021), SciTePress, pp. 274–281. doi:10.5220/0010681400003064. 3

[MHG10] Malik M. M., Heinzl C., Groeller M. E.: Comparative Visualization for Parameter Studies of Dataset Series. *IEEE Transactions on Visualization and Computer Graphics 16*, 5 (2010), 829–840. doi:10.1109/TVCG.2010.20. 3

[Mic23] Microsoft: Monaco editor - a browser based code editor. https://github.com/microsoft/monaco-editor, 2023. Accessed: 2023-11-30. 8

[MLW*19] Muller M., Lange I., Wang D., Piorkowski D., Tsay J., Liao Q. V., Dugan C., Erickson T.: How Data Science Workers Work with Data: Discovery, Capture, Curation, Design, Creation. In *ACM CHI Conference on Human Factors in Computing Systems* (2019), pp. 126:1–126:15. doi:10.1145/3290605.3300356. 6

[Mye86] Myers E. W.: AnO(ND) difference algorithm and its variations. *Algorithmica 1*, 1 (1986), 251–266. doi:10.1007/BF01840446. 3, 6

[NAK21] Nassar A. H., Adib E., Kwiatkowski D. J.: Distribution of KRASG12C Somatic Mutations across Race, Sex, and Cancer Type. *New England Journal of Medicine 384*, 2 (2021), 185–187. doi:10/ghvp8t. 8

[NHM20] Nugroho Y. S., Hata H., Matsumoto K.: How different are different diff algorithms in Git? *Empirical Software Engineering 25*, 1 (2020), 790–823. doi:10.1007/s10664-019-09772-z. 3

[NSH*17] Niederer C., Stitz H., Hourieh R., Grassinger F., Aigner W., Streit M.: TACO: Visualizing Changes in Tables Over Time. *IEEE Transactions on Visualization and Computer Graphics 24*, 1 (2017), 677–686. doi:10.1109/TVCG.2017.2745298. 4, 6

[OJEF19] Ondov B., Jardine N., Elmqvist N., Franconeri S.: Face to Face: Evaluating Visual Comparison. *IEEE Transactions on Visualization and Computer Graphics 25*, 1 (2019), 861–871. doi:10.1109/TVCG.2018.2864884. 4

[PC05] Pirolli P., Card S.: The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of International Conference on Intelligence Analysis* (VA, USA, 2005), vol. 5, McLean, pp. 2–4. 2

[Per18] Perkel J. M.: Why Jupyter is data scientists' computational notebook of choice. *Nature 563*, 7729 (2018), 145–146. doi:10.1038/d41586-018-07196-1. 2

[Per21] Perkel J. M.: Reactive, reproducible, collaborative: computational notebooks evolve. *Nature 593*, 7857 (2021), 156–157. doi:10.1038/d41586-021-01174-w. 2, 10

[PMBF19] Pimentel J. F., Murta L., Braganholo V., Freire J.: A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)* (2019), pp. 507–517. doi:10.1109/MSR.2019.00077. 2

[PZK*22] Psallidas F., Zhu Y., Karlas B., Henkel J., Interlandi M., Krishnan S., Kroth B., Emani V., Wu W., Zhang C., Weimer M., Floratou A., Curino C., Karanasos K.: Data Science Through the Looking Glass: Analysis of Millions of GitHub Notebooks and ML.NET Pipelines. *ACM SIGMOD Record 51*, 2 (2022), 30–37. doi:10.1145/3552490.3552496. 2

[Res23] Resemble.js: Image Analysis and Comparison. https://github.com/rsmbl/Resemble.js, 2023. Accessed: 2023-11-30. 3, 4

[RSBB23] Ramasamy D., Sarasua C., Bacchelli A., Bernstein A.: Visualising data science workflows to support third-party notebook comprehension: an empirical study. *Empirical Software Engineering 28*, 3 (2023), 58. doi:10.1007/s10664-023-10289-9. 3, 4

[RTH18] Rule A., Tabard A., Hollan J. D.: Exploration and Explanation in Computational Notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal, Canada, 2018), ACM, pp. 1–12. doi:10.1145/3173574.3173606. 2, 3, 5, 8, 10

[Sch21] Schmidt J.: Visual Data Science. In *Data Science, Data Visualization, and Digital Twins*, Sara Shirowzhan, (Ed.). IntechOpen, Rijeka, 2021. doi:10.5772/intechopen.97750. 2

[SGB13] Schmidt J., Gröller M. E., Bruckner S.: VAICo: Visual Analysis for Image Comparison. *IEEE Transactions on Visualization and Computer Graphics 19*, 12 (2013), 2090–2099. doi:10.1109/TVCG.2013.213. 4

[SHGC18] Sutton C., Hobson T., Geddes J., Caruana R.: Data Diff: Interpretable, Executable Summaries of Changes in Distributions for Data Wrangling. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (New York, USA, 2018), KDD '18, Association for Computing Machinery, pp. 2279–2288. doi:10.1145/3219819.3220057. 4

[SHH99] Studholme C., Hill D. L. G., Hawkes D. J.: An overlap invariant entropy measure of 3D medical image alignment. *Pattern Recognition 32*, 1 (1999), 71–86. doi:10.1016/S0031-3203(98)00091-0. 7

[SKR18] Samuel S., König-Ries B.: ProvBook: Provenance-based Semantic Enrichment of Interactive Notebooks for Reproducibility. In *International Semantic Web Conference (ISWC) Demo Track 2018* (2018). URL: http://ceur-ws.org/Vol-2180/paper-57.pdf. 2, 3

[Tan23] Tang A.: @armantang/html-diff - generate html content diffs. https://github.com/Arman19941113/html-diff, 2023. Accessed: 2023-11-30. 8

[WBSS04] Wang Z., Bovik A., Sheikh H., Simoncelli E.: Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (2004), 600–612. doi:10.1109/TIP.2003.819861. 7

[WDBD21] Weinman N., Drucker S. M., Barik T., DeLine R.: Fork It: Supporting Stateful Alternatives in Computational Notebooks. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (New York, USA, 2021), CHI '21, Association for Computing Machinery. doi:10.1145/3411764.3445527. 2

[WEDD22] Wang A. Y., Epperson W., DeLine R. A., Drucker S. M.: Diff in the Loop: Supporting Data Comparison in Exploratory Data Analysis. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New York, USA, 2022), CHI '22, Association for Computing Machinery, pp. 1–10. doi:10.1145/3491102.3502123. 4

[WKLZ20] Wang J., KUO T.-Y., Li L., Zeller A.: Assessing and Restoring Reproducibility of Jupyter Notebooks. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (2020), pp. 138–149. URL: https://ieeexplore.ieee.org/document/9286024. 2

[WZC*19] Wenskovitch J., Zhao J., Carter S., Cooper M., North C.: Albireo: An Interactive Tool for Visually Summarizing Computational Notebook Structure. In *2019 IEEE Visualization in Data Science (VDS)* (2019), pp. 1–10. doi:10.1109/VDS48975.2019.8973385. 2