REUSING INTERACTIVE ANALYSIS WORKFLOWS

Kiran Gadhave
University of Utah

Zach Cutler
University of Utah Alexander Lex
University of Utah
alex@sci.utah.edu

ABSTRACT

Interactive visual analysis has many advantages, but has the disadvantage that analysis processes and workflows cannot be easily stored and reused, which is in contrast to scripted analysis workflows using a programming language such as Python. In this paper, we introduce methods to semantically capture workflows in interactive visualization systems for different interactions such as selections, filters, categorizing/grouping, labeling, and aggregation. We design these workflows to be robust to updates in the dataset by capturing the semantics of underlying interactions, and, hence, they can be applied to updated datasets. We demonstrate this specification using a prototype that visualizes the data, shows interaction provenance, and allows generating workflows from this provenance. Finally, we introduce a Python library that can consume the workflow and apply it to the datasets, providing a seamless bridge between computational workflows and interactive visualization tools. We demonstrate our techniques using our UI prototype and Jupyter notebooks.

Keywords Visualization, reusing workflows, interaction, provenance

1 Introduction

Data visualization enables analysts to leverage the powerful human visual system to identify patterns and draw conclusions. When data visualizations are made interactive with selections and data transformations such as filters, labels, and aggregation, they can reveal complex relationships and make large datasets accessible. Not suprisingly, therefore interactive data visualization has entered the mainstream, with tools like Tableau and Microsoft PowerBI, but also many more specialized visual analysis platforms seeing widespread use and commercial success.

One significant drawback of interactive visual data analysis, however, is that analysis processes remain ad hoc: when a dataset is updated or changed, the analysis has to be re-done. Updating datasets is very common: For example, business add new sales data regularly, scientists expand or correct their datasets as errors are discovered or new samples come in, and economists get updated data about various countries' indicators every year. However, datasets that are updated are problematic for data transformations that are used in a downstream analysis: in such cases, even storing the state of an application is not enough to meaningfully reapply a filter.

This lack of reusability in interactive visual analysis is in sharp contrast to computational analysis workflows. A function that filters a dataset based on parameters can be easily reapplied to an updated dataset. However, this application comes with the usual drawbacks of computational approaches: they require a skilled analyst, are harder to write, and cannot leverage the benefits of graphical perception.

In this paper, we propose methods to capture and reuse workflows in an interactive visualization system. Workflows are composed of interactions that are well suited for interactive data visualizations, such as selecting, filtering, labelling, categorizing, or aggregating items. We introduce methods to capture these workflows in a semantically meaningful way, making them robust to changes in the underlying datasets, as illustrated in Figure 1.

However, even when we use semantically meaningful selection to reapply them to a new dataset, we need human review — and potentially updates — to ensure that these actions reflect the analyst's intent. To address this, we introduce a review and update interface that ensures that changes are correctly applied and makes corrections easy.

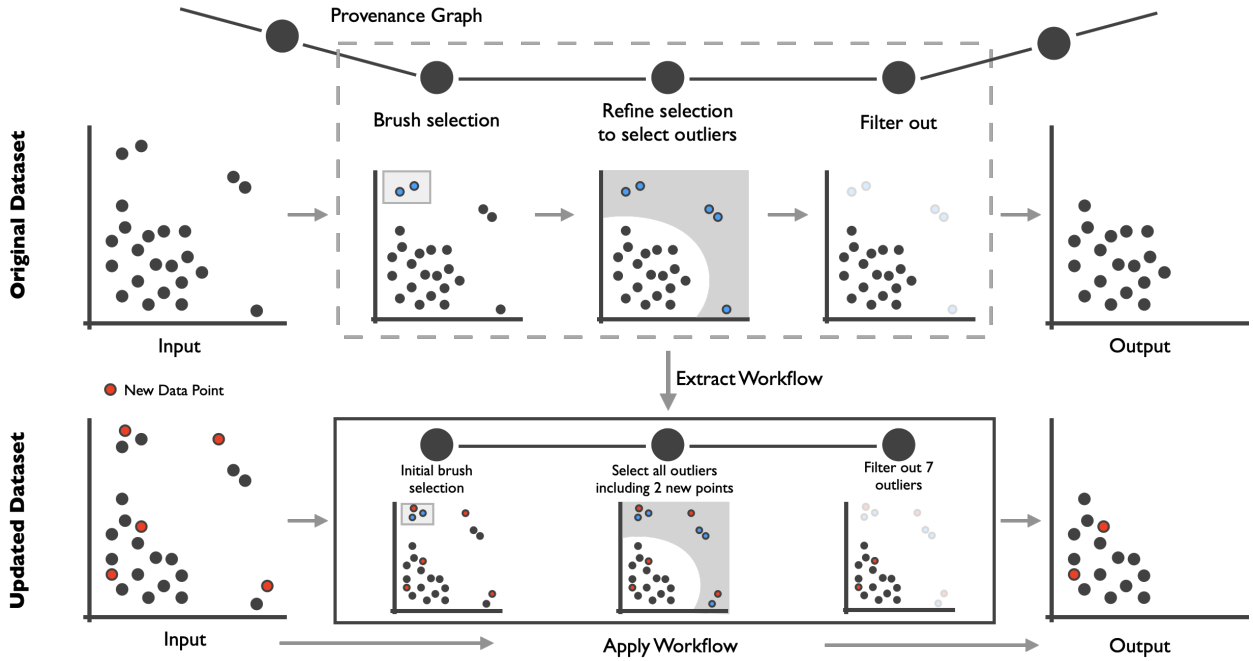

Figure 1: The process of reusing workflows. Actions, such as brushes or filters, are applied on a dataset. Specifying a pattern-based intent — such as outliers in this example — improves reusability. A series of actions can be extracted into a workflow. This workflow can then be applied to an updated dataset.

Finally, capturing smart workflows has the advantage that we can make them available within the same visualization system and use them to bridge between interactive visualization systems and scripted data analysis processes. For example, an analyst could do some preprocessing in a Jupyter notebook, then launch an interactive visualization system from the notebook to execute a series of complex selections and data transformations that are more easily achieved in a visualization system, and then return to the notebook to apply, e.g., an algorithm to the transformed dataset. Because we now have reusable visualization workflows, all parts of such an analysis can then be reapplied to an updated dataset.

We demonstrate these capabilities in a prototype visualization system that captures interaction provenance, from which analysts can extract workflows. We demonstrate that these workflows can be reapplied to updating or changing datasets on a series of examples. We also introduce a Python software library that can be used to bridge between the visualization system and Python code, and provide examples for these workflows.

In summary, our contribution is a method to capture workflows in interactive visualization systems that can then be reapplied to a new dataset. To ensure the accuracy of the reapply process, we introduce review and updating capabilities for these workflows. Finally, we introduce methods to use these workflows as part of computational workflows. We believe that our methods will make it possible to use interactive visualizations even for analyzing datasets that are being updated. We also push the limits of what is possible with regard to integration between computational and interactive workflows, thereby enabling analysts to leverage the best tool for each part of a job.

2 Related Work

Our work is related to methods for capturing interaction, to approaches for managing analysis workflows, and to the integration between interactive visualization techniques and computational workflows. We discuss all of them separately in this section.

2.1 Capturing Interaction

Despite widespread agreement in the visualization literature about the benefit of interaction, it is often treated as secondary to the development of visual encodings [1, 2] or even as an afterthought in the design of visualization systems [3]. Several authors also call for a new "science of interaction" [4, 3] to support the human reasoning process.

Interaction is used in a variety of ways in a visual data analysis session, including choosing datasets, selecting items, making data transformation, and zooming and panning. To be able to capture interactions, reflecting on the types of interactions that are commonly used in visual analysis systems is useful. Although various taxonomies of interaction have been devised (e.g., [5, 6, 1, 7]), we will use the taxonomy introduced by Heer and Shneiderman [8] that distinguishes among three high-level categories of "interactive dynamics": (1) **data and view specification**,

with the sb-categories visualize (specify data and visual encoding), filter, sort, and derive; (2) **view manipulation**, with the subcategories select, navigate, coordinate (synchronize between multiple views), organize (arrange windows and workspaces), and (3) **process and provenance** (record, annotate, share, guide). Tracking the "data and view specification" operations is highly relevant to enable reuse, whereas the importance of tracking certain "view manipulation" operations depends on the system, task, and dataset. The third category, "process and provenance," is a collection of meta-interactions that we want to enable in this work.

Data-aware selections, or, more generally, data aware actions (selections, queries, filters, etc.), are defined in *data space* [9, 10, 8]. For a selection, for example, this means it is described by conditions, not by a list of items. Dynamic queries [11] are commonly realized in a data-aware way: all items that fit certain conditions, defined e.g., via sliders, are considered to be in the query results. Certain types of selections (brushes) [12] can be realized in a data-aware way, using, for example, a rectangular selection in a scatterplot that easily translates into the necessary conditions. More often, however, selections (and other actions) are realized by direct reference, e.g., by pointing at items, and, hence, they are defined in *item space*. Actions that are defined in item space have several disadvantages: they cannot be generalized to apply to updating data, and they cannot be used to semantically explain a selection. Data-aware actions, in contrast, are robust to changes, can be used to explain and justify an action, and can be used in various ways to support an analyst, e.g., by relaxing a selection [13], or for reuse in a different context [14].

Most data-aware selections are realized by deriving rules directly from a brush. Such selection, however, is limited to cases where spatial position is used to encode attributes in visualization techniques such as scatterplots or parallel coordinates. In more general cases, rules for data-aware selections are harder to derive. It is possible, however, to derive the pattern of a selection (what makes the item in a selection belong to each other and different from everything else) algorithmically. Xiao, Gerth, and Hanrahan, for example, create "knowledge representations" of selections in communication networks [15], and Su, Paris, and Durand use a similar approach for selecting graphical objects [14]. Most relevant to our work is the framework to infer pattern-based intents of a selection by Gadhave et al. [16], which we utilize as the backbone of our approach to reusing actions. This framework makes it possible to capture the semantics of selections, which we leverage as part of our approach to capturing and reapplying workflows.

2.2 Workflows

Explicit modeling of workflows is common in scientific data analysis [17]. Representative examples are systems such as Galaxy [18] for biomolecular data, SCIRun [19] and Kepler [20] for scientific/simulation data, and KNIME [21] in a machine learning context. Workflow approaches are also common for scientific visualizations applications such as volume rendering. Here, VisTrails [22] is a prolific example. Notable workflow-based systems for abstract data visualization include GraphTrail [23], where each node in the workflow shows an aspect of a multivariate network, and VisFlow [24], which is tailored to tabular data.

Explicitly modeled workflows are designed to be easily reused. At the same time, the definition of these workflows is similar to explicit code-based specification of visualizations, and thus the associated interaction cost [2] is high, and the spontaneity and rapid exploration that is associated with interaction patterns such as direct manipulation [25] is lost. Also, even if they are easier to learn than writing code, they have a steeper learning curve compared to interactive systems.

An alternative approach to explicit workflow modeling is tracking the interaction provenance [26, 27] and using this information to later extract workflows. Although several visualization systems track provenance [28, 29, 30, 31] and a few dedicated libraries to track provenance exist [32, 33], most tools do not explicitly curate workflows based on the provenance. A notable exception is the Vistories tool [34], which enables analysts to curate data stories. However, these data stories cannot be reused on different datasets.

Chen et al. proposed a parameteric symbolic approach to support analytic provenance in their CZSaw system [35]. CZSaw enables analysts to reuse parts of the analysis process, but the process is reapplied based on a parametric model created earlier. The system does not support autodetection and application of patterns, and the analysis has to be done in the same system.

Commercially available solutions like Tableau Prep make it possible to create workflows for preparing and cleaning data. The interactions supported by Tableau Prep are combining data, filling in missing values, etc. Tableau prep does not support creating a workflow visually by interacting with the data directly, or exporting it to other environments.

2.3 Integrating Visualization and Computational Environments

Computational notebook-based environments are great for narrative data analysis, fulfilling Knuth's vision of literate programming [36]. A common limitation of notebook platforms like Jupyter has been a lack of interactive visualizations to transform the data. Native visualization libraries, such as Matplotlib [37], have limited interactive capabilities, and cannot feed back actions from visualizations to code. Tableau views can be embedded in Jupyter notebooks, but the visualizations have to be developed in Tableau environment and can be embedded only in Jupyter for display. Libraries such as Altair [38] support interactive visualizations, but the interactions primarily serve the purpose of coordinating between multiple views, not for

data transformation. Schmidt and Ortner [39] discuss a number of reasons for the lack of interactive data analysis in notebook-style environments, with two of the reasons being the lack of computational power and limited interaction capabilities native to the environment. Our approach aims to bridge the gap between these environments by providing communication between the computational notebook and the interactive visualization tool, where the latter can fill in for lack of interactivity in the former.

Wu et al. [40] introduce B2, a set of techniques that treat the data queries as a shared representation between code and visualizations. They introduce a library that adds a visualization dashboard in a Jupyter notebook. This dashboard can interact with Python code. Our approach develops a similar shared representation, but instead leverages an external visualization tool, which enables the use of more sophisticated visualization systems. Most importantly, however, in contrast to B2, selections and derived data transformations in our approach are captured on a semantic level, and hence are robust to changes in datasets.

3 Capturing and Reusing Workflows

In this section, we describe the details of how we capture workflows in interactive visualizations that can then be reapplied to updated datasets. We first introduce the types of interactions we capture, and then explain how we capture workflows, followed by how workflows can be reused with updated datasets. We conclude the section by showing how we can use our approach to bridge between interactive visualizations and computational workflows. We demonstrate our approach using a simple, multiple-coordinated-view interactive visualization systems using scatterplots and parallel coordinates, shown in Figure 2. A live demo of our tool is available at `https://reapply-workflows.github.io/reapply-workflows/`.

3.1 Interactions

We developed a specification to capture interaction provenance that we use to capture the workflows. The actions we capture and support in our visualization system are informed by the taxonomy developed by Heer and Shneiderman [8]: we support all of the "data & view specification" actions except for sorting, which is meaningful only in tabular visualizations, as well as all "process & provenance" interactions. However, in our specification we capture the interactions that are relevant for data manipulation, namely *view specification*, *selections*, and various types of *data manipulation*.

View specification interactions are concerned with choosing the subset of dimensions to visualize. An example is to show two dimensions of a dataset in a scatterplot: here, the view specification entails both the choice of dimensions and the choice of visualization technique.

Selections are a basic but very important interaction available in visualizations. Selections not only allow highlighting items of interest, but also form the basis for further data transformation on selected subsets of the data. For our specification, we break down selections by the level of semantics each type of selection captures:

The simplest of these is *ID-based selection*, which directly stores the IDs of the selected items. ID-based selection has the lowest level of semantics and is the least useful when reusing the selection: as only IDs of items are stored, if items are added in a new version of a dataset, they could not be considered, even if they clearly fall into a selected pattern.

Next we specify the *range selection* that stores ranges over dimensions, capturing a set of rules for a selection, similar to e.g., an SQL query. Range selections are usually specified using rectangular brushes [12, 9]. They are reusable for updates in the data as long as the updates happen within the extent of the range selection. For example, if an updated version of the dataset has three new points within a rectangular brush area, these points will be selected automatically.

The highest level of selection specification is *pattern-based selections* as introduced by Gadhave et al. [16]. This approach captures higher level semantics behind the selection that are apparent when the data is visualized. A pattern-based selection recognizes, for example, that an analyst selected all outliers, or a cluster centered at a specific location. By selecting based on higher level patterns in the data, such selections are robust to changes in the data: For example, when outliers are selected in a dataset, similar outliers can be selected in an updated dataset, even if the outliers appear in new locations.

Our specification supports four **data transformation** actions, shown in Figure 3. *Filter* actions track whether certain items are filtered-in or filtered-out of the dataset. Filters is useful for focusing on a select group of items by filtering them in or removing irrelevant data items by filtering them out. *Labeling* sets of items in a dataset is useful for annotating or tagging items with metadata or observations. *Categorize* actions are used to classify (assign categories) to items from a set of dynamically defined categories. Usually, each point is assigned to a unique category. Categorization is useful for dividing the dataset into distinct subsets that can then be compared or used separately in subsequent analysis steps.

Deriving new data items by *aggregating* groups of existing ones is an important data transformation, as is evident from the popularity of pivot tables in Excel. An aggregate item can replace the items it was derived from, simplifying the data. Aggregation also allows analysts to compare groups with shared characteristics effectively. Aggregation is done by grouping multiple data items into a new item. Each attribute is aggregated based on a mathematical function (often called "apply" function in programming libraries). These functions usually summarize multiple values into a single representative value like the *mean*, *median*, *sum*, *min*, or *max* of the item's attribute, but custom functions

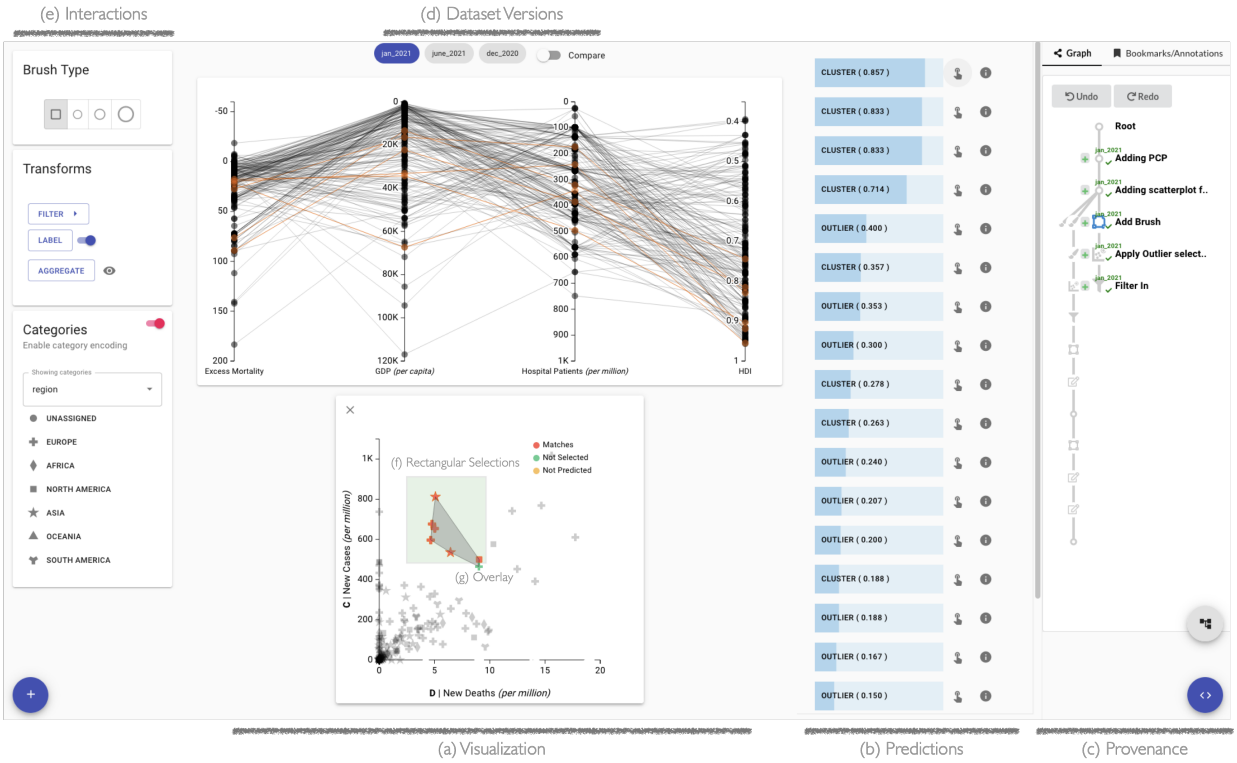

Figure 2: Prototype visualization showing a COVID-19 dataset. The analyst selected six dimensions to be shown in a scatterplot and a parallel coordinate plot (a). (f) A rectangular brush selection is used to compute predictions for patterns to capture the semantics of the selection. (b) A ranked list of these predictions is shown to the right of the scatterplot. The analyst selects the top prediction, which is a cluster, and the system shows an overlay (g) to show the the boundary of the cluster. (c) The provenance graph on the right shows the captured interactions.

for aggregation are also common. In a dataset on metrics about countries, for example, it might be useful to aggregate all European countries into a single "Europe" item, and compare it to an "Asia" item. For this aggregation, we have to define "population" of Europe as the sum of the population of all the countries in it. A column like "life expectancy", however, would need a different, more sophisticated function for meaningful aggregation.

We have chosen these interactions as they allow us to demonstrate our approach, yet other operations, such as deriving new attributes, or manipulation, such as moving around items, could be equally supported by our methods. Taken together, these view specification, selection, and data transformation actions make up a powerful set of tools that benefit strongly from being available in an interactive visualization interface and are commonly used in data science tasks.

3.2 Capturing Workflows

We define a workflow as a user-curated series of interactions or operations executed in the course of a visual data analysis session. Figure 4 shows an example worklfow that selects outliers, filters them in (so that only the outliers remain), and then assigns categories to some of them based on a selection. One of our design principles was to make the process of capturing workflows easy, with minimal overhead for the user. For this reason, we chose to enable analysts to extract workflows based on provenance data after they executed a series of actions [41]. As a result, analysts can freely explore and interact with a dataset, and only worry about capturing workflows after they have successfully completed an operation.

We track interactions and store them in a directed acyclic graph using the Trrack library [33]. Branches of the history are possible when users go back to a previous step and continue an analysis from that point. We visualize the provenance graph in a tree-like layout (Figure 4(a)), where each action is described and can be annotated by the user. We track all types of interactions that are needed to reconstruct every state of the application.

The key to reusable workflows is to capture semantically meaningful selections, as downstream data transformations are built on top of the selections. In our method, **selections** are captured on multiple levels as described in Section 3.1. These different selection approaches can — and commonly are — combined. A rough selection can be made by a rectangular selection, which is then augmented by adding a

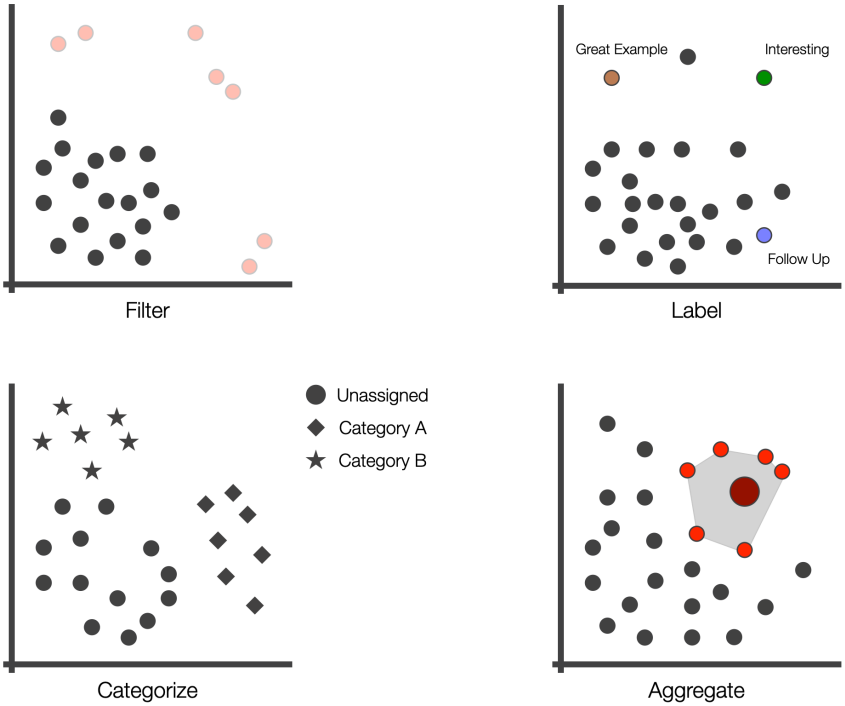

Figure 3: Overview of the data transformations supported by our system: **Filter** removes selected items, **Label** assigns a (possibly) unique label to an item, **Categorize** classifies items into categories, and **Aggregate** reduces a set of items to one derived item (in this example, the light-red items are aggregated into the dark-red, larger item).

few items. Analysts can then use a pattern-based selection, derived from the selection they just made.

Capturing a selection by ID and range is relatively straightforward. However, capturing semantics is more challenging. Gadhave et al. [16] define pattern-based selection intents as "the reasoning behind selections based on statistical patterns or structures in a dataset." Figure 5 illustrates this concept for four patterns — in/outliers, clusters, correlations, and multivariate optimization (also called "skylines"). The upper row shows these patterns with a selection (blue) of elements in the pattern. By capturing these pattern-based intents, we can reapply the action even on updated datasets, as illustrated in the bottom row. All these patterns have changed slightly (values changed; items added or removed), yet the patterns remain clearly visible, and a selection based on these patterns preserves them.

We use Gadhave et al.'s approach to capture these semantically rich selections: Our prototype monitors user-selections and compares them to a large set of patterns that were precomputed for a given dataset using various algorithms and parameterizations. The different patterns are ranked based on the Jaccard Similarity between the selection and the prediction, as shown in Figure 2(b). Here, we see a rectangular selection that partially covers a cluster, and the system ranks a clustering pattern as a good match. When an analyst hovers over the cluster prediction, the extent of the cluster is shown as a polygon, and the items that are not part of the selection are highlighted. When an analyst chooses to confirm this prediction as the intended pattern, our system stores the details of this pattern.

We predict five different patterns: clusters, in- and outliers, correlations, multivariate optimization, and ranges. For each of these patterns, we store the information necessary to recreate the pattern in an updated dataset. For example, for clustering, we store which type of algorithm was used (e.g., KMenas or DBScan) with which parameters, in addition to attributes about the specific cluster that is selected, such as its centroid.

Data transformations are derived from selections: to categorize a group of points, for example, they are first selected and then assigned to a category. The robustness of the transformation depends on how the selection was achieved. For example, if an analyst selects 15 items individually and assigns them to a category, the category is associated with just those items. However, if the items were selected using a range selection, the category is associated with the range rather than individual items. When the dataset is updated, items appearing in the region of the range selection are categorized as well. Using semantic selections further improves robustness of subsequent actions. An analyst can use the pattern-based selection to refine the initial selection of 15 items as a cluster. The category is then associated with the cluster rather than the original selection. Updating the dataset by adding or removing items automatically updates the categorization as well. If the groups of items were to move, the semantic selection would still accurately

Figure 4: The workflow editor. An analyst can create a new workflow from the interaction history captured in (a), the provenance graph, and curate it in (b), the workflow editor interface by removing unnecessary actions. (c) Previously created workflows are shown on the right and can be loaded and edited at any time.

track the items, in contrast to a range selection, which would loose items that move outside its range.

Figure 6 shows an example where an analyst first added a plot of a dataset that exhibits clusters. The analyst then continued with a crude rectangular selection. The system recommends a cluster as a match in the prediction interface. Hovering over that cluster prediction reveals the cluster's properties. The analyst decides this is a good match for the intended selection and confirms the prediction. Finally, the analyst filters out the selected items. Each of these steps is then reflected in the provenance graph. The analyst could now go on and continue a selection, and revisit the filter process at a later time. However, the analyst can also create a new workflow form the provenance history and call it "Remove Outliers". This workflow can then be reused for a new dataset.

3.3 Reusing Workflows

When reusing workflows, we apply a workflow captured through a visual analysis to an updated or changed dataset. Tabular datasets can change in a limited number of ways: attributes associated with rows can change, and rows can be added or removed. Also, dimensions or rows (items) can be added, removed, or reordered [42]. For our purposes, we limit ourselves to changes of existing attributes, adding or removing items, and updating attribute values, as order is relevant only for certain representations and adding or removing of dimensions is beyond the scope of our work. Figure 7(a) shows how we visualize these changes. Analysts can select two out of several versions of a dataset and study their relationships.

When a new version of a dataset is loaded, we by default apply all actions in the current history to the new dataset.

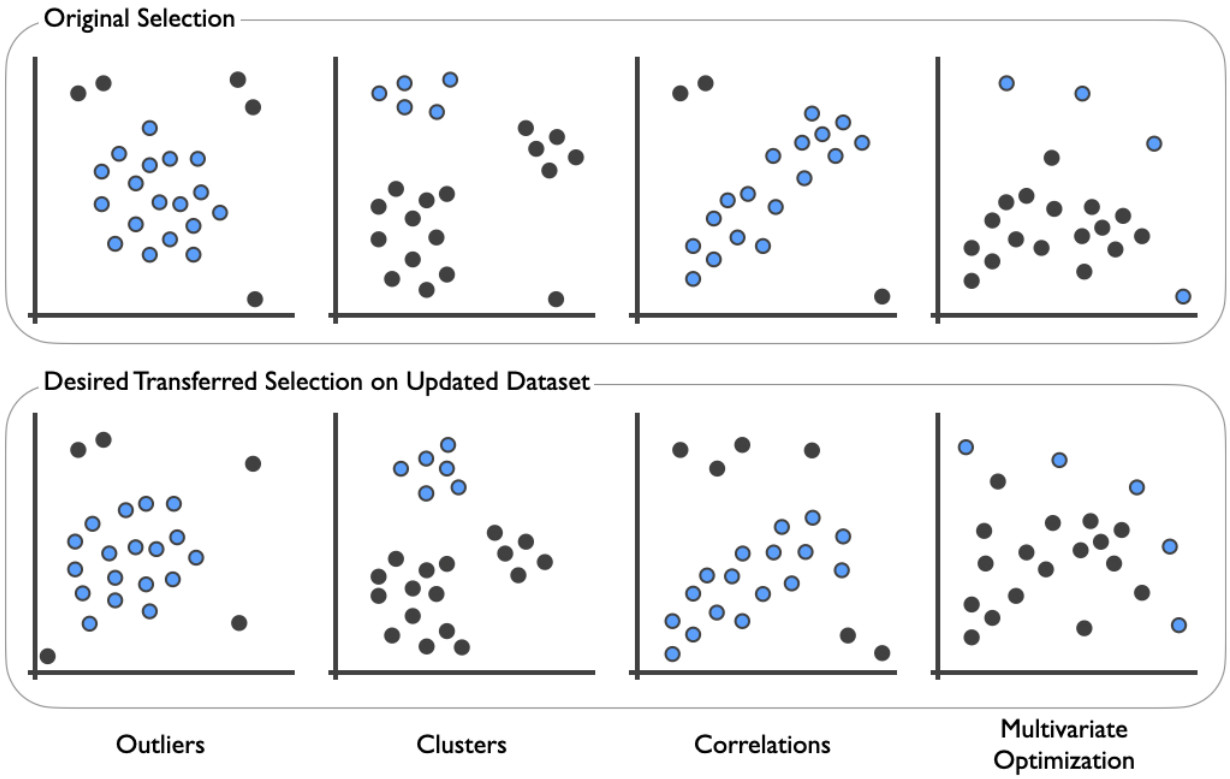

Figure 5: Example selections based on pattern-based intents. Selections are made based on in/outliers, clusters, correlations, and multivariate optimization (skyline) patterns, respectively. By capturing the semantics of these selections, we can reapply the selection to datasets that are changed in various ways (items moved, added, or removed), as shown in the second row.

We have different approaches to apply selections, which is straightforward for IDs and ranges. The method for reapplying semantically captured selections varies by the type of selection. Common to these approaches is re-running the specific algorithm used to predict the pattern with the captured parameters. The clustering algorithms (KMeans and DBScan) return multiple clusters as output; hence, we have to match up a specific cluster to a selection. For KMeans, in addition to the parameters, we also store the centroid of the selected cluster. After re-running KMeans with the same parameters on an updated dataset, we find the cluster with the centroid closest to the centroid stored in the selection. For DBScan, we identify the best-fit cluster based on the Jaccard similarity between the items in the selected cluster and the cluster produced by the re-run of DBScan. For patterns like multivariate optimization and regression, we store meta information about the selected prediction such as the "sense" (the direction of the optimization) for the multivariate optimization, and the location of selected items for linear regression such as "within" or "outside" the regression threshold.

Figure 7 shows a comparison between the dataset shown in Figure 6 and an updated dataset, and a subsequent successful application of a cluster based selection. We can see in Figure 7(a) how the dataset changed compared to the one in Figure 6. Figure 7(b) shows how the selected cluster changed between the dataset; the hulls of both clusters are shown. Figure 7(c) shows the cluster selection in the updated dataset. If the initial selection had been captured using range-based rules, i.e., not using a semantic pattern, the items that shift outside the range would not have been captured.

Figure 8 shows a more difficult example, where the selected and subsequently filtered cluster broke apart into two clusters. Depending on the intent of the analyst, several options are plausible: remove both clusters, remove only the top or bottom cluster, or remove none of the clusters. Here, an automatic determination is impossible, as the right action depends on the analyst's higher level intent. Hence, the analyst has to review this situation, and make a decision on how to proceed, as illustrated in Figure 8. By default, the system assumes that one large cluster is the best match (Figure 8(a)), as it best corresponds to the previously selected cluster overall. Assuming that the analyst intends to select only the top, smaller cluster, they can reject the "Clusters Selection" node in the provenance graph, and replace it with a better matching cluster, as shown in Figure 8(b).

Figure 6: Brushing a dataset that exhibits clusters. The analyst roughly selects a cluster and then uses the predicted pattern-based selections to explore and refine the selection.

3.4 Bridging to Computational Workflows

The final aspect that capturing semantically meaningful workflows enables us to do is to deeply integrate workflows with computational analysis systems and re-execute a workflow on an updated dataset just like a function. Figure 9 shows how we bridge between the **visualization tool** and the **computational environment**. The piece that connects the visualization tool and the computational environment is a **workflow database**. An analyst can work in the visualization tool to perform a visual analysis and capture workflows, as discussed before. Whenever a workflow is created or modified, it is also stored in a workflow databases. The workflows can then be loaded from the workflow database to a computational environment, such as a Jupyter notebook.

Figure 10 shows the process of loading and using a workflow from the workflow database. We provide a library that interfaces with the workflow database to provide convenient access. This library provides functions to print descriptions and inspect the steps a workflow takes. Ultimately, workflows can be applied to a pandas dataframe containing the data. The output of applying the workflow depends on the type of workflow itself. If the workflow is made only of selections, the output dataframe has an extra boolean column *isSelected* that denotes the selected items. More complex workflows with data transformation have similar modifications to the output dataframe, e.g., filters return a subset of the original dataset after executing the filter, label and categorize workflow has an extra column with the relevant label or category assignment, and aggregation workflows add a new row to the dataset with the aggregated values.

When a workflow is applied to an updated dataset in a computational environment, the library executes the same operations to find the best match for actions based on the semantic selections as our web-tool does (see Section 3.3). When a workflow is used on an updated dataset, the library prints a warning, informing users that this workflow has not been reviewed for that version of the dataset. It also prints a link to the visualization tool that automatically loads the updated dataset with the workflow applied to it. Analysts can now visually verify and possibly refine the workflow, and then continue their analysis in the computational environment.

4 Prototype Implementation

To demonstrate our technique for capturing and reapplying the workflow, we developed a prototype that

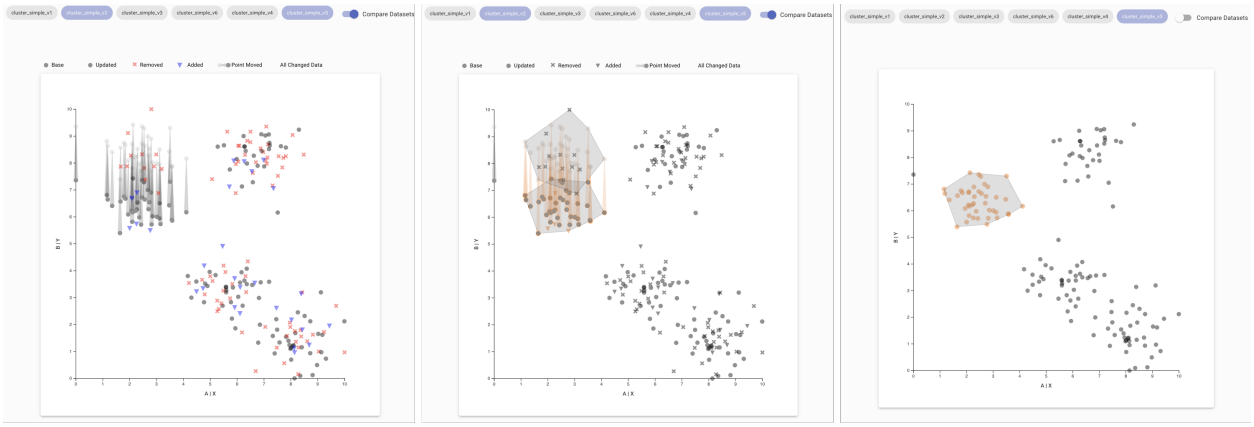

Figure 7: Comparing two datasets and reapplying a workflow. (a) The comparison mode explicitly shows the differences between two selected versions of a dataset. The scatterplot encodes newly added items in the updated dataset as blue triangles ▼, removed items are shown as red crosses ✖, and items that have shifted positions show a comet-like trail from their original to their new position. (b) we can see that the selection made on the original dataset nicely moves down to the new cluster and handles new and removed items correctly. (c) The updated selection.

allows interactive visual analysis and a library that can reapply the workflows. The prototype is available at `https://reapply-workflows.github.io/reapply-workflows/`.

Users may create their own projects, and then upload as many datasets as they wish to a project. Any datasets uploaded via the tool will also be available for analysis in the library. For the purpose of demonstrating our tool, we have prepared several demonstration datasets with multiple versions.

Analysts may select any two dimensions of a tabular dataset to view in a **scatterplot**, and may add as many plots as desired. We provide rectangular and free-form "paint-brushes" of three sizes for selection. Every time a selection is made, predictions are immediately updated. Predictions are visualized by highlighting the items that are inside (or outside) the prediction, and also by an explicit visualization of the pattern, such as a cluster outline (see Figure 7) or a visualization of the Pareto frontier for multivariate optimization (see Figure 13).

Analysts may add a **parallel coordinate plot** to visualize multiple dimensions at once. We provide brushing on each axis to select items. The predictions are updated as in the scatterplot for every selection. Predictions are visualized by highlighting the items that are inside or outside the prediction.

Analysts can freely switch between different versions of a dataset, and any analysis done on a previous dataset will automatically be applied to the new dataset. Although having unrelated datasets in a project is not enforced, the datasets within a project should be related, otherwise transferring actions might not make sense.

Our prototype visualization tool is implemented using React, D3 and Typescript. The back-end uses a Flask server and an PostgreSQL database. The workflow database is stored in a Firebase Realtime Database, and communicates with the visualization tool and the library using the Firebase-Admin API.

4.1 Reapply Library

At the heart of our method is the **Reapply Library** that performs all the predictions and the matching of actions between updated datasets. The library is used in both the web-based tool and third-party notebooks. The library is written in Python and supports Python versions greater than 3.7. The library uses scikit-learn to run the prediction algorithms.

The library loads a workflow from the workflow database using the workflow ID. Analysts can use the *describe* method to check the workflow before applying it to a dataset. Analysts can call the *apply* method with the target dataframe and label column to apply the workflow, which in turn returns a result object. We show our computational demos in Google Colab notebooks, which are equivalent to Jupyter notebooks but can be collaboratively edited and are hosted by Google.

5 Analysis Examples

We validate our approach through a series of examples with a mix of artificial and real datasets. We show how our approach can be used to create reusable workflows for various interactions.

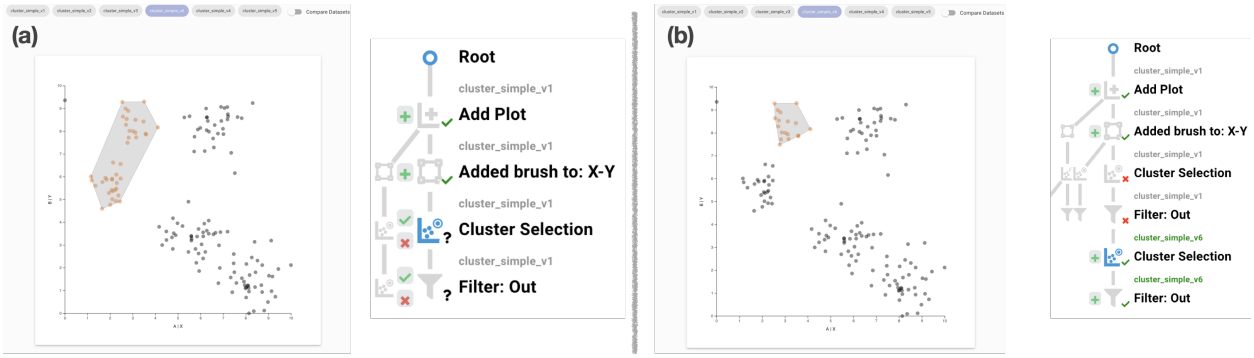

Figure 8: Reviewing and updating workflows that were applied to an updated dataset. Continuing our analysis from Figure 6 the analyst loaded a new version of the dataset where a part of a cluster broke off and moved down. (a) By default, the system considers these two clusters to be one larger cluster, since they are still relatively close, and the previously selected larger cluster biases the outcome toward a single cluster. The review interface indicates that certain actions have not been reviewed for this version of the dataset, by showing a question mark (?) next to the node. To select just the upper cluster, the analyst first confirms the "Add Plot" and "Added Brush" actions, which are then show as approved with a check-mark (✔). (b) The analyst then rejects the "Cluster Selection" action and pick a new cluster prediction that captures their intent.

5.1 Categorizing and analyzing outlier countries for COVID-19

For this example, we are using COVID-19 data provided by the "Our World in Data" project [43]. This dataset includes various COVID-19- related metrics for multiple countries across the world. We took subset of these attributes for selected months for our demos. COVID-19 data attributes change frequently and are a good way to demonstrate our approach, since selections and conclusions must be robust to updates in data.

Let us look at a scenario for which the analyst wants to investigate countries that have aberrant trend in number of new cases and number of new deaths related to COVID-19. We start with data for January 2021 and load a scatterplot for **new monthly cases** vs **new monthly deaths**.

We immediately see that many countries are far away from the cluster of countries close to the origin. We then select a few of these countries using a paint brush selection. The system computes predictions and suggests an outlier-based selection that selects all the outliers (see Figure 11(a)). We use this suggestion to refine our selection. We switch to different months of the dataset to see if the selection is applied correctly. We are happy with the selection, so we filter-in these items to focus on these outliers.

We then categorize these outliers based on new monthly cases per million. We select all the countries with high monthly cases and high monthly deaths with a rectangular brush and categorize them as countries with "High Deaths–High Cases". We then select countries with low monthly cases but high monthly dates and categorize them as countries with "High Deaths–Low Cases", and finally select countries with low monthly deaths but high monthly cases and categorize them as "Low Death–High Cases" (Figure 11(b). We switch to different datasets and verify that the categorization is applied correctly. When we are satisfied with the result, we approve the interactions in the provenance history. Figure 11(c) shows an extreme example, June 2021, where cases and deaths in most countries are much lower, clustering close to the origin of the chart. Applying the categories (Figure 11(d)) results in several outliers being unassigned, hinting at the fact that even moderate COVID activity is an outlier in that version of the dataset.

We now curate the provenance history into a **Categorize Outliers** workflow and store it to the workflow database. We then move to a Jupyter notebook to load this workflow and analyze these newly categorized countries. We can create a histogram of the categorized countries stacked by the region to get a general idea about how different regions were affected by COVID 19, where we see that high deaths have shifted to South American countries in June, which were barely affected in January.

5.2 Analyzing the Relationship of GDP and Child Mortality

For this example, we will use a global health dataset that is updated yearly. Let us look at a scenario where we want to investigate countries that have high child mortality rates. We load in data from the Gapminder project [44] for the year 2010 in the visualization tool. We add a scatterplot of *child mortality rate* (CMR) vs *gross domestic product* (GDP), an indicator of a countries wealth. We immediately see a strong trend in the data: most countries with a high child mortality rate have low GDP, but there are also many countries with low GDP and low CMR.

To focus on countries with high CMR, we use the rectangular brush from CMR values from 100–200. Our system suggests a simplified range selection to generalize the ini-

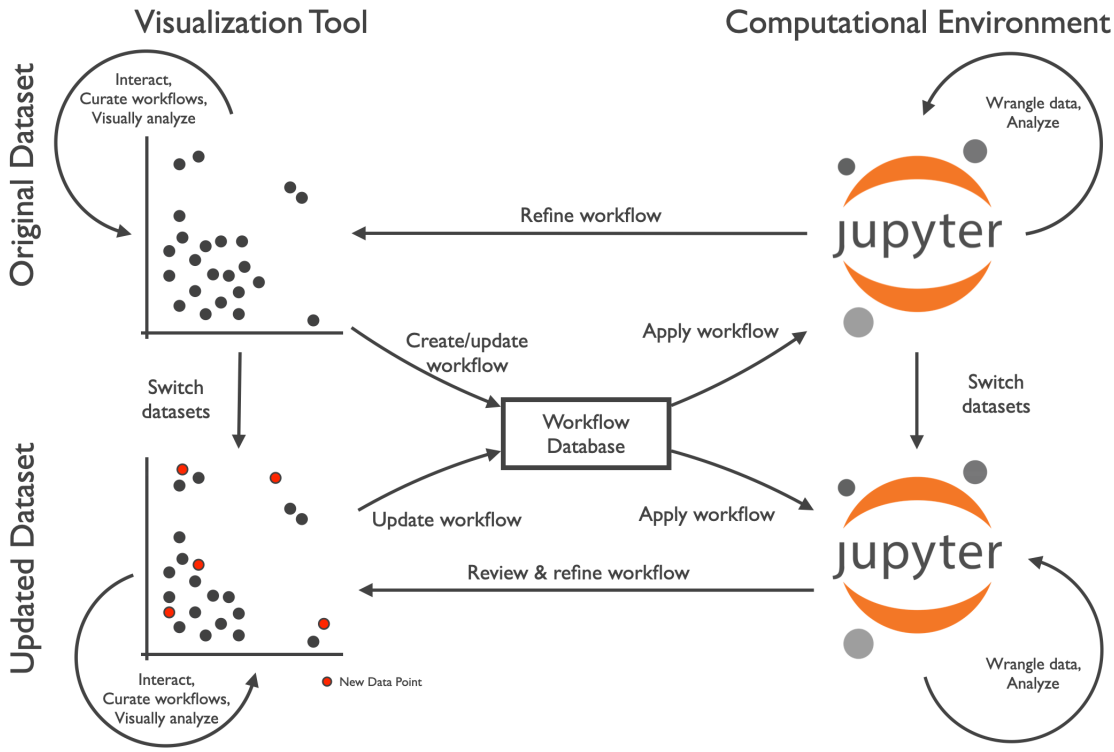

Figure 9: Analysis flow when bridging between a visualization tool, where workflows can be curated, and a computational environment, where workflows can be (re-)applied. Workflows created in the visualization tool are synced with a workflow database. Computational environments, such as a Jupyter notebook, have access to the workflows, and can apply it to the dataset by simply executing a function. When a dataset is updated in the computational environment, the workflow can be applied to the new datasets, with results immediately available. Analysts can also review and potentially refine the updated workflow in the visualization tool. If the workflow is updated in this way, it is then immediately available in the computational environment.

tial brush (see Figure 12(a)), which also includes Haiti — the rightmost point — with a value of 209.

We now decide to look at some historical data for CMR, and switch to the data for 1980. The system reapplies the interactions, and the generalized range selection includes the new countries with high CMR automatically (see Figure 12(b)). We are satisfied with the generalization and confirm the interactions in the provenance graph for the 1980 data (see Figure 12(c)).

We can now curate these interactions into a workflow, store it in the workflow database, and move over to a Jupyter notebook to continue the investigation to look for reasons of high child mortality. We can use this workflow to also automatically filter out the items of interest for other years in the dataset in the Jupyter notebook.

5.3 Other Patterns and Updated Datasets

In figure 13, we demonstrate how our methods can be applied to different patterns, and how these patterns are adapted when a changed dataset is loaded. We show an original dataset and a pattern-based selection, a comparison of how the data changed, and finally how the selection was applied to the changed dataset. For outliers, we can see that the points that move closer to the center are excluded from the outlier selections, and the points that move away are added to the outliers. For the range selection, we see the system added and removed the points that fall outside the rules for the range selection. For multivariate optimization, the system updates the selection on the new dataset, and we see an updated Pareto frontier (shown as a staircase). For correlations, we show a linear regression line and threshold within which the points are counted as part of the correlation. The system updates the selection based on points moving in and out of the threshold.

6 Discussion

Certainty of Fit for Reuse. When we apply a selection to a new dataset, we currently assume that an analyst will review the update selection. Although a review is certainly necessary if the data changed significantly, minor changes might not require a manual review. We could conceivably compute metrics about how "sure" we are about a specific operation, as it is applied to a new dataset. If, for example, all points are in the same selection and have moved little,

Figure 10: Executing a computational workflow defined in the visualization tool in a computational environment. (a) We first load the dataset. (b) We load the workflow library and the particular workflow we are interested in. We then apply the workflow to the dataset. The tool plots a preview of the actions. Note that new `isSelected` and `isFiltered` Boolean column are introduced when a brush and filter are added in the preview. (c) After the filter is applied, the number of rows is reduced from 150 to 108. A visualizations of the result shows the cluster was removed. Visit the the notebook.

we might not need a review. If, in contrast, the dataset has changed significantly and the selection is affected, we could print a warning, emphasizing the need for a review.

Higher Dimensional Patterns. Our examples were restricted to relationships between a small number of dimensions. In practice, the computational parts of our approach work just as well on higher dimensional patterns. However, selections of higher dimensional patterns are difficult in scatterplots, and only slightly easier in parallel coordinate plots. To address this problem, we plan on integrating projection plots (e.g., t-SNE or PCA), which could serve as the user-interface of the selection.

Other Data Types. Our implementation and our choice of algorithms is specific to tabular data,but our general approach is not. We could equally apply our methods to network data, image data, or volumetric data, provided we can identify suitable algorithms to semantically capture selections.

Interaction Directly in Notebooks. Visualization libraries such as Altair [38] and B2 [40] have made interactive selections in visualizations within a Jupyter notebook possible. We plan on extending our library, so that selections made within a notebook can also be autocompleted and extracted into a workflow. While we expect that other aspects, such as compound actions and reviewing of workflows, are infeasible to integrate natively within a notebook, robust, pattern-based selections would enhance an analyst's ability to leverage the interactive capabilities of such simple visualizations.

Reapplying to Unrelated Datasets. Our methods for transferring actions to updated datasets are robust up to a point. The clustering case in Figure 8 shows situations in which the automatic transfer does not succeed: when a pattern changes so much that a different interpretation is possible. While we remedy these situations through our review process, it would be worthwhile to automatically make alternative suggestions on which actions could be

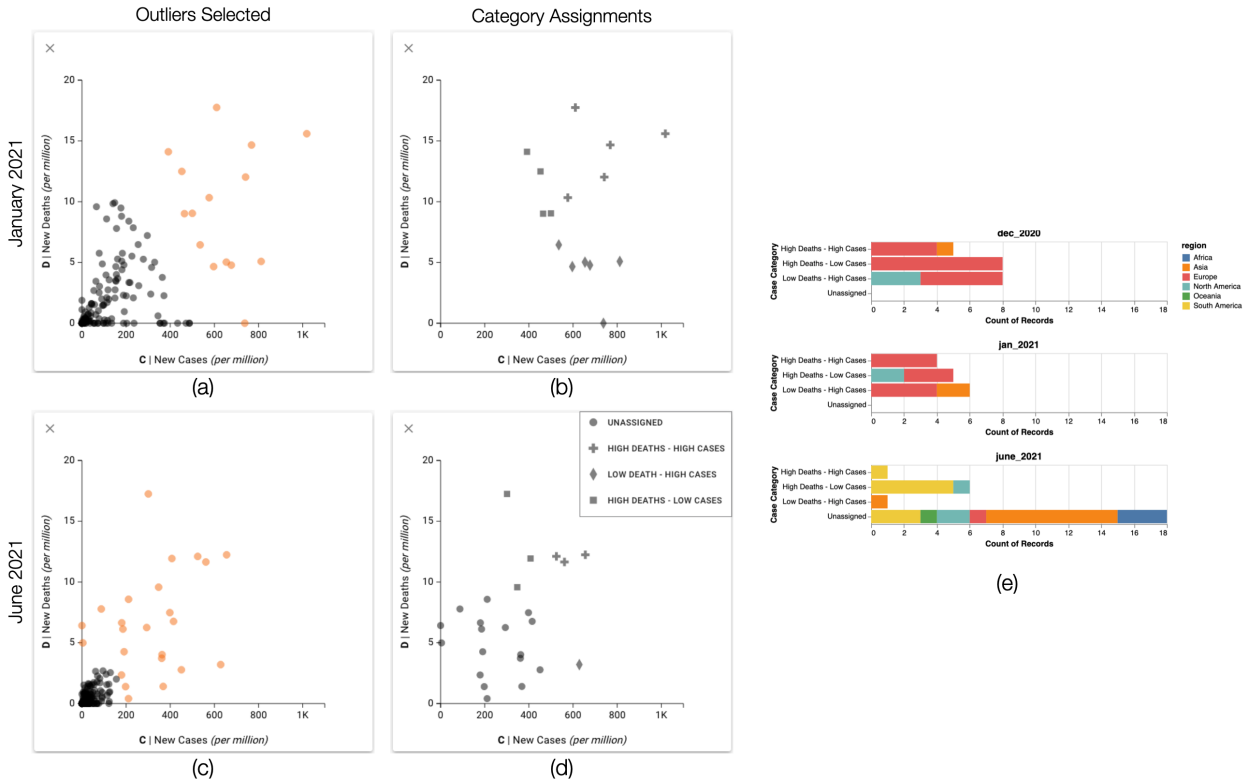

Figure 11: Categorization of countries in a COVID-19 dataset. (a) A scatterplot for January 2021 shows the number of new cases per million inhabitants vs. the number of new deaths per million. An analyst has selected outliers, capturing countries with many cases, many deaths, or both. (b) The analyst filters out all countries except the selected outliers, and categorizes the countries based on new cases values into "High Deaths–High Cases" ✚, "High Deaths–Low Cases" ■, and "Low Deaths–High Cases" ♦ categories. (c) We switch to an updated dataset for June 2021 and the analyst sees that the number of cases and deaths have gone down across the world, but the pattern-based selection has correctly selected the outliers. (d) The system automatically applies the range-based categorization to the countries that fall within the ranges in June 2021. (e) A subsequent analysis in a notebook reveals that the worst of the pandemic has shifted to South America in June 2021. Visit the interactive version of this figure, or the corresponding notebook.

taken. We have not experimented with applying workflows to altogether different datasets. Our current technique of capturing the workflow relies on tracking view specifications and downstream selections and transforms. Applying the dataset to unrelated dataset will almost always result in incorrect results. However, we see potential in using workflows as templates for recurring tasks, such as data cleanup on datasets generated by the same instrument although for different experiments. Here, a human analyst could update the parameters of a selection, or supplement it, but use a subsequent data transformation.

Making Differences Explicit. In some situations, it would be useful to be able to retrieve explicit results of which items have been added to or removed from our selection. In the child mortality case, described in Section 5.2, for example, it would be useful to learn which countries have moved from the high child mortality range to the low mortality range.

7 Conclusion

We have introduced a method to capture interactive actions taken in a visualization in a semantically meaningful way and to reuse sequences of actions (workflows) on updated datasets. In this way, we make actions taken in a visualization just as robust to changes as if they were implemented in a function in code. We introduce methods that match up selections between updated datasets that go beyond just reapplying a simple rule, instead leveraging various pattern-detection algorithms and knowledge about the properties of a prior selection. We introduce a mechanism to review changes and update workflows if necessary. Finally, we have demonstrated that this approach also allows us to bridge between an interactive visualization system and a computational workflow.

While any of the selections we capture through interactive selections could also be implemented directly in code, we argue that our approach is easier to execute, especially for analysts with limited knowledge about the various algo-

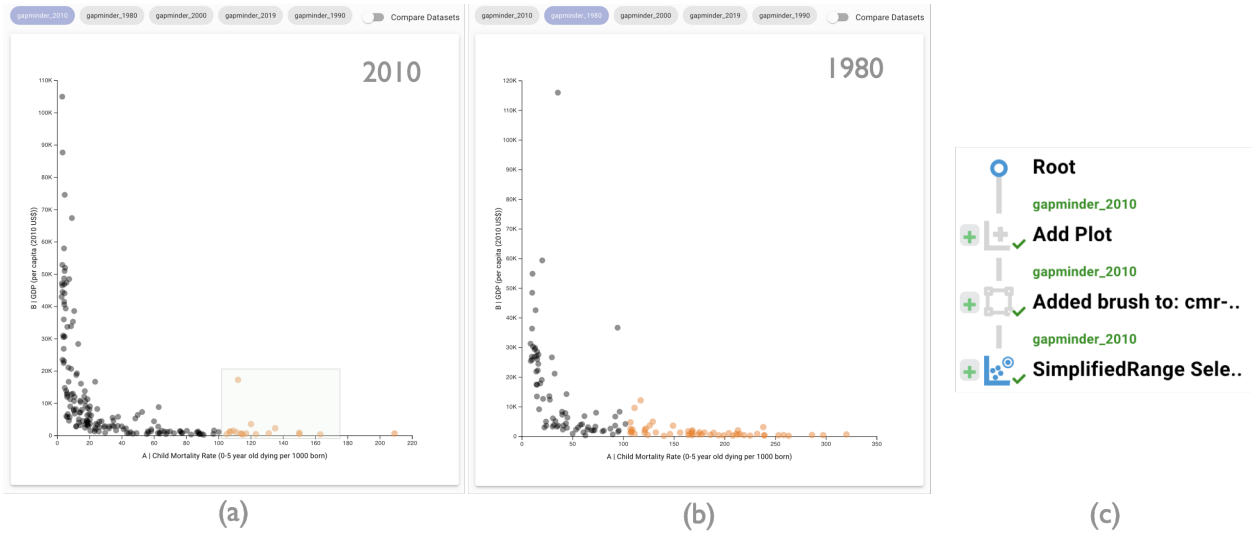

Figure 12: Preserving a selection in a temporally changing public health dataset. (a) We make a selection of countries with high child-mortality. We missed an extreme outlier on the far right, but the suggested simplified range selection catches that mistake. (b) Applying this brush to the 1980 dataset preserves the semantics of the actions, although now significantly more countries are selected (note the scale change). (c) The provenance graph for this sequence of actions.

rithms we employ. Extracting some common data transformation interactions as workflows for frequently updated datasets makes it easier to get started with a new version. Our prototype and our examples show that our approach works for a range of patterns and for datasets that change in significant ways. We believe that our approach could also be transferred to many other types of data and types of visualizations.

References

[1] Ji Soo Yi, Youn ah Kang, John Stasko, and J.A. Jacko. Toward a Deeper Understanding of the Role of Interaction in Information Visualization. IEEE Transactions on Visualization and Computer Graphics (InfoVis '07), 13(6):1224–1231, 2007.

[2] H. Lam. A Framework of Interaction Costs in Information Visualization. IEEE Transactions on Visualization and Computer Graphics, 14(6):1149–1156, November 2008.

[3] William A. Pike, John Stasko, Remco Chang, and Theresa A. O'Connell. The Science of Interaction. Information Visualization, 8(4):263–274, 2009.

[4] James J. Thomas and Kristin A. Cook. Illuminating the Path: The Research and Development Agenda for Visual Analytics. IEEE Computer Society Press, 2005.

[5] Ben Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In Proceedings of the IEEE Symposium on Visual Languages (VL '96), pages 336–343, 1996.

[6] Leland Wilkinson. The Grammar of Graphics. Springer, second edition, 2005.

[7] David Gotz and Michelle X. Zhou. Characterizing Users' Visual Analytic Activity for Insight Provenance. Information Visualization, 8(1):42–55, 2009.

[8] Jeffrey Heer and Ben Shneiderman. Interactive dynamics for visual analysis. Communications of the ACM, 55(4):45–54, 2012.

[9] Allen R. Martin and Matthew O. Ward. High Dimensional Brushing for Interactive Exploration of Multivariate Data. In Proceedings of the IEEE Conference on Visualization (Vis '95), pages 271–278. IEEE Computer Society Press, 1995.

[10] Mark Derthick, John Kolojejchick, and Steven F. Roth. An Interactive Visual Query Environment for Exploring Data. In Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology, UIST '97, pages 189–198, New York, NY, USA, 1997. ACM.

[11] B. Shneiderman. Dynamic queries for visual information seeking. IEEE Software, 11(6):70–77, November 1994.

[12] Richard A. Becker and William S. Cleveland. Brushing Scatterplots. Technometrics, 29(2):127–142, 1987.

[13] Jeffrey Heer, Maneesh Agrawala, and Wesley Willett. Generalized Selection via Interactive Query Relaxation. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '08, pages 959–968, New York, NY, USA, 2008. ACM.

[14] Sara L. Su, Sylvain Paris, and Frédo Durand. QuickSelect: History-based Selection Expansion. In Proceedings of Graphics Interface 2009, GI '09, pages 215–221, Toronto, Ont., Canada, Canada, 2009. Canadian Information Processing Society.

[15] Ling Xiao, J. Gerth, and P. Hanrahan. Enhancing Visual Analysis of Network Traffic Using a Knowledge Representation. In Visual Analytics Science And Technology, 2006 IEEE Symposium On, pages 107–114, 31 2006-nov. 2.

[16] Kiran Gadhave, Jochen Görtler, Zach Cutler, Carolina Nobre, Oliver Deussen, Miriah Meyer, Jeff M. Phillips, and Alexander Lex. Predicting intent behind selections in scatterplot visualizations. Information Visualization, 20(4):207–228, October 2021.

[17] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. Future Generation Computer Systems, 25(5):528–540, May 2009.

[18] Jeremy Goecks, Anton Nekrutenko, James Taylor, and T. Galaxy Team. Galaxy: A comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol, 11(8):R86, 2010.

[19] Steven G. Parker and Christopher R. Johnson. SCIRun: A Scientific Programming Environment for Computational Steering. In Proceedings of the ACM/IEEE Conference on Supercomputing (SC '95), page 52. ACM, 1995.

[20] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows. In Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004., pages 423–424, June 2004.

[21] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Kilian Thiel, and Bernd Wiswedel. KNIME - the Konstanz Information Miner: Version 2.0 and Beyond. SIGKDD Explor. Newsl., 11(1):26–31, November 2009.

[22] Louis Bavoil, Steven P. Callahan, Carlos Scheidegger, Huy T. Vo, Patricia Crossno, Claudio T. Silva, and Juliana Freire. VisTrails: Enabling Interactive Multiple-View Visualizations. In Proceedings of the IEEE Conference on Visualization (VIS '05), pages 135–142, 2005.

[23] Cody Dunne, Nathalie Henry Riche, Bongshin Lee, Ronald Metoyer, and George Robertson. GraphTrail: Analyzing Large Multivariate, Heterogeneous Networks While Supporting Exploration History. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12), pages 1663–1672. ACM, 2012.

[24] B. Yu and C. T. Silva. VisFlow - Web-based Visualization Framework for Tabular Data with a Subset Flow Model. IEEE Transactions on Visualization and Computer Graphics (InfoVis '16), 23(1):251–260, 2017.

[25] B. Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. Computer, 16(8):57–69, August 1983.

[26] Chris North, Remco Chang, Alex Endert, Wenwen Dou, Richard May, Bill Pike, and Glenn Fink. Analytic Provenance: Process+Interaction+Insight. In CHI '11 Extended Abstracts on Human Factors in Computing Systems, CHI EA '11, pages 33–36, 2011.

[27] Kai Xu, Alvitta Ottley, Conny Walchshofer, Marc Streit, Remco Chang, and John Wenskovitch. Survey on the Analysis of User Interactions and Visualization Provenance. Computer Graphics Forum, 39(3):757–783, 2020.

[28] Jeffrey Heer, Jock Mackinlay, Chris Stolte, and Maneesh Agrawala. Graphical Histories for Visualization: Supporting Analysis, Communication, and Evaluation. IEEE Transactions on Visualization and Computer Graphics (InfoVis '08), 14(6):1189–1196, 2008.

[29] M. Kreuseler, T. Nocke, and H. Schumann. A History Mechanism for Visual Data Mining. In Proceedings of the IEEE Symposium on Information Visualization (InfoVis '04), pages 49–56, 2004.

[30] Yedendra B. Shrinivasan and Jarke J. van Wijk. Supporting the analytical reasoning process in information visualization. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '08, pages 1237–1246, 2008.

[31] Stef van den Elzen and Jarke J. van Wijk. Small Multiples, Large Singles: A New Approach for Visual Data Exploration. Computer Graphics Forum (EuroVis '13), 32(3pt2):191–200, 2013.

[32] Akhilesh Camisetty, Chaitanya Chandurkar, Maoyuan Sun, and David Koop. Enhancing Web-based Analytics Applications through Provenance. IEEE Transactions on Visualization and Computer Graphics, 25(1):131–141, January 2019.

[33] Zachary T Cutler, Kiran Gadhave, and Alexander Lex. Trrack: A Library for Provenance Tracking in Web-Based Visualizations. In IEEE Visualization Conference (VIS), pages 116–120, Salt Lake City, UT, USA, 2020. IEEE.

[34] Samuel Gratzl, Alexander Lex, Nils Gehlenborg, Nicola Cosgrove, and Marc Streit. From Visual Exploration to Storytelling and Back Again. Computer Graphics Forum, 35(3):491–500, 2016.

[35] Yingjie Victor Chen, Zhenyu Cheryl Qian, Robert Woodbury, John Dill, and Chris D. Shaw. Employing a Parametric Model for Analytic Provenance. ACM Transactions on Interactive Intelligent Systems, 4(1):6:1–6:32, April 2014.

[36] D. E. Knuth. Literate Programming. The Computer Journal, 27(2):97–111, January 1984.

[37] J. D. Hunter. Matplotlib: A 2D Graphics Environment. Computing in Science Engineering, 9(3):90–95, May 2007.

[38] Jacob VanderPlas, Brian E Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. Altair: Interactive statistical visualizations for Python. Journal of open source software, 3(32):1057, 2018.

[39] Johanna Schmidt and Thomas Ortner. Visualization in Notebook-Style Interfaces. In Proceedings of the Workshop on the Gap between Visualization Research and Visualization Software (VisGap), May 2020.

[40] Yifan Wu, Joseph M. Hellerstein, and Arvind Satyanarayan. B2: Bridging code and interactive visualization in computational notebooks. In ACM User Interface Software & Technology (UIST), 2020.

[41] E.D. Ragan, A. Endert, J. Sanyal, and J. Chen. Characterizing Provenance in Visualization and Data Analysis: An Organizational Framework of Provenance Types and Purposes. IEEE Transactions on Visualization and Computer Graphics (VAST '15), 22(1):31–40, 2016.

[42] Christina Niederer, Holger Stitz, Reem Hourieh, Florian Grassinger, Wolfgang Aigner, and Marc Streit. TACO: Visualizing Changes in Tables Over Time. IEEE Transactions on Visualization and Computer Graphics (InfoVis '17), 24(1):677–686, 2017.

[43] Hannah Ritchie, Edouard Mathieu, Lucas Rodés-Guirao, Cameron Appel, Charlie Giattino, Esteban Ortiz-Ospina, Joe Hasell, Bobbie Macdonald, Diana Beltekian, and Max Roser. Coronavirus Pandemic (COVID-19). Our World in Data, March 2020.

[44] Hans Rosling and Zhongxing Zhang. Health advocacy with Gapminder animated statistics. Journal of Epidemiology and Global Health, 1(1):11, 2011.

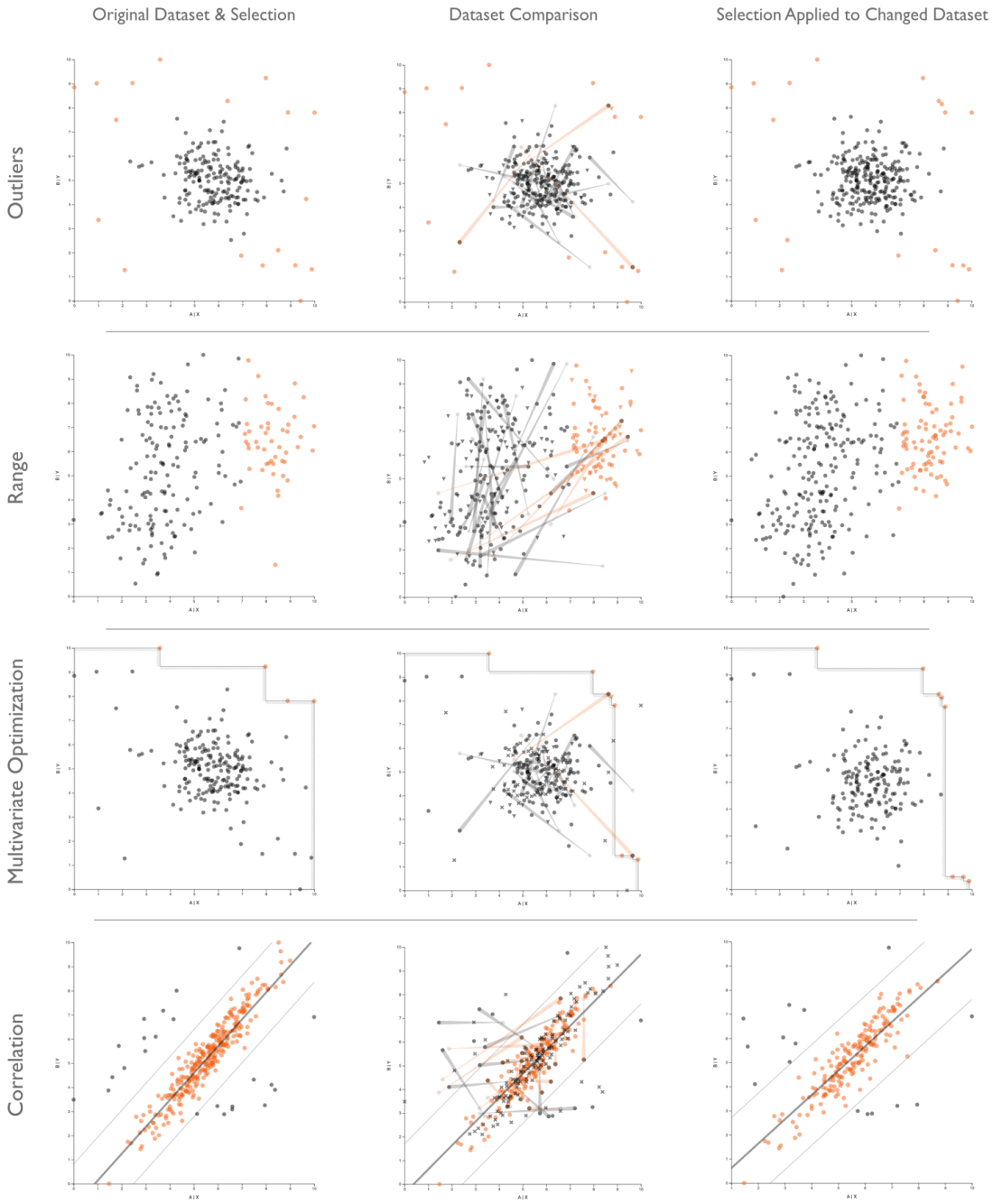

Figure 13: Semantic reapplication of different patterns in synthetic datasets. We see how different patterns (rows) are semantically applied to updated datasets. The first column is the original selection, the second column shows the compare view between the original dataset and the updated dataset, and the third column shows the final selections on the updated dataset.