Data Debugging and Exploration with Vizier

Mike Brachmann[β], Carlos Bautista[n], Sonia Castelo[n], Su Feng[i], Juliana Freire[n],
Boris Glavic[i], Oliver Kennedy[β], Heiko Müller[n], Rémi Rampin[n], William Spoth[β],
Ying Yang[o]

β: University at Buffalo n: NYU i: IIT o: Oracle
{ mrb24, okennedy, wmspoth }@buffalo.edu sfeng14@hawk.iit.edu bglavic@iit.edu
{ carlos.bautista, s.castelo, juliana.freire, heiko.mueller, remi.rampin }@nyu.edu
ying.y.yang@oracle.com

ABSTRACT

We present Vizier, a multi-modal data exploration and debugging tool. The system supports a wide range of operations by seamlessly integrating Python, SQL, and automated data curation and debugging methods. Using Spark as an execution backend, Vizier handles large datasets in multiple formats. Ease-of-use is attained through integration of a notebook with a spreadsheet-style interface and with visualizations that guide and support the user in the loop. In addition, native support for provenance and versioning enable collaboration and uncertainty management. In this demonstration we will illustrate the diverse features of the system using several realistic data science tasks based on real data.

ACM Reference Format:
Mike Brachmann, Carlos Bautista, Sonia Castelo, Su Feng, Juliana Freire, Boris Glavic, Oliver Kennedy, Heiko Müller, Rémi Rampin, William Spoth, Ying Yang. 2019. Data Debugging and Exploration with Vizier. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3299869.3319887

1 INTRODUCTION

The unprecedented scale at which information is being produced today has lead to wide-spread use of data-centric methods in industry, science, government, and every-day life. These methods heavily rely on the availability of high-quality data and on products derived from such data, e.g., machine learning models and visualizations. The creation of a data product is a complex process that involves multiple tasks including ingest, curation, exploration, and debugging. Commonly users start with an initial, imperfect data pipeline, and iteratively refine and fix this pipeline until a desired result has been achieved. Notebook frameworks such as Jupyter (https://jupyter.org/) have become popular tools for creating data products. They allow users to craft documents that interleave documentation, scripts, and outputs (e.g., datasets or plots). Similarly, less tech-savvy users use spreadsheets for the same purpose. Both paradigms (notebooks and spreadsheets) mix code with outputs to provide immediate feedback on the effect of a change to the code or data. For instance, when a user edits cells in a spreadsheet all plots visualizing the data are updated automatically or when a user runs a python script in a notebook cell to replace missing values and subsequently re-executes a cell creating a plot, then the updated plot will reflect the updated data. In summary, the interactive nature of spreadsheets and notebooks makes them powerful tools for data product creation. Unfortunately, existing notebook and spreadsheet software suffer from limitations which severely impede users in their work with data. **Vizier** (https://github.com/VizierDB), the system we present in this demonstration, overcomes these limitations. Vizier is a data centric, multi-modal, provenance-aware platform for data curation, exploration, debugging, and analysis.

1.1 Requirements

Before explaining the shortcomings of the aforementioned paradigms, we will first motivate a list of requirements for systems that support users in developing data pipelines.

R1. Supports Iterative Construction of Pipelines: The system should aid users in the iterative process of creating a pipeline by making outputs of operations available and refreshing them automatically when the pipeline is modified. Furthermore, provenance should be made available for the

SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3319887

whole history of a pipeline such that users can work with multiple versions and can backtrack to a working version if the pipeline has evolved into a non-productive direction.

R2. Expressiveness + Ease-of-use: The system should allow users to manually curate data, to process it using scripts, and to query it. The interaction paradigm provided for that should be both expressive enough as well as simple to use.

R3. Aids Data Debugging: Raw data is almost never directly suited for creating high-quality data products and analysts and data scientists spend the majority of their time on data discovery and curation. It is critical to support users in these complex, time-consuming, and error-prone processes.

R4. Scalability: Many data products, even if of small size themselves, are based on large datasets necessitating support for scalable operations. Furthermore, certain interaction paradigms are not suited well for large datasets, e.g., it is impossible to manually fix errors in multi-billion row datasets.

R5. Traceability and Provenance: Consumers of data products would want to know whether the products are trustworthy which requires provenance to be tracked.

R6. Collaboration and Reuse: It should be possible for multiple users to collaborate on a data pipeline and data product. Users should be able to adapt existing pipelines to new versions of a dataset or similar use cases.

1.2 Notebooks/Spreadsheet Limitations

Given these requirements, we now explain how notebook and spreadsheet systems fall short in several regards.

Limitations of Notebooks. Existing notebooks are optimized for manipulating data through bulk, scripted transformations and organize those pipelines into cells. However, dependencies among datasets and pipeline cells are not tracked. Thus, there is no provenance tracking for the cells and no automatic refresh for pipeline outputs when data or pipelines are updated. This makes notebooks hard to debug and hard to construct iteratively, especially if they contain non-trivial dependencies among cells [5], because the user has to be aware of all cell dependencies to decide what cells to re-execute and in which order when the notebook is updated. Furthermore, this hinders reproducibility because a user that is not the creator of the notebook has to analyze the code of all cells to understand in which order the cells have to be run. Finally, this also leads to hard to trace bugs when the outcome changes because cells were run in an order other than the intended one. Data exploration and curation tasks often benefit from manual data entry or other spreadsheet data manipulation operations which are hard to implement

[1]while there exist spreadsheet widgets for Juptyer, e.g., https://github.com/QuantStack/ipysheet they are not bound to datasets as in Vizier.

[2]while approaches like [2] are efficient, the paradigm of manual updates that is fundamental to spreadsheets does not scale in terms of human effort.

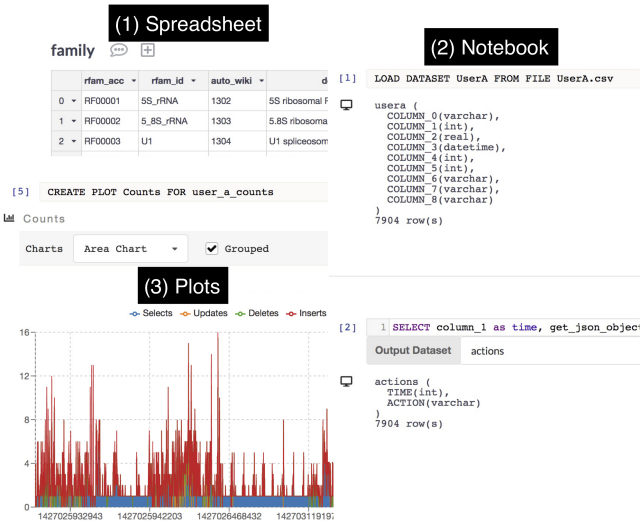

Figure 1: Vizier's multi-modal user interface combining spreadsheets, notebooks and plots.

using notebook interfaces. Also, the data being manipulated is opaque, making manual error fixing cumbersome. Furthermore, due to the lack of dependency tracking, notebooks do not scale well in case of complex pipelines. Since existing notebook systems do not support collaboration and versioning natively, users rely on external version control systems such as git for collaboration. Thus, the decision of when to create a version is left to the user which can lead to versions getting lost, e.g., a user forgot to commit a change.

Limitations of Spreadsheets. Spreadsheets are suited best for manual data entry and updates. Thus, they can be quite effective for fixing heterogeneous errors in small datasets (one-off-fixes). However, this type of operations does not scale to larger datasets and is not suited for fixing large numbers of similar errors or for detecting errors automatically. Spreadsheets do not provide any means to structure operations into a pipeline as well as provenance support. Hence, any data pipeline consisting of multiple steps is hard to implement, modify, and debug with spreadsheet software. Pipelines that have been painstakingly created for one dataset are hard to generalize to another dataset because all edits are manual and specific to a particular piece of data.

2 SYSTEM OVERVIEW

Figure 2 shows the high-level architecture of the Vizier software stack. Our demonstration will focus on Vizier's Web User Interface (Web-UI), a web-based frontend that we will discuss shortly. The Web-UI (and other user-facing interfaces) interact with Vizier through a REST API (Web-API) with endpoints for creating, viewing, and manipulating Vizier notebook workflows. The Web-API also provides paginated access to datasets resulting from these workflows, as

Req.	Spreadsheets	Notebooks	Vizier
R1	(+) interactive nature (-) lack of controlflow structure (-) no build-in versioning	(+) interactive nature (+) organizes pipeline into cells (-) dependencies among datasets and cells are hidden [5] (-) no automatic refresh on update (±) no build-in versioning, users commonly rely on VCS such as git	(+) interactive nature (+) automatic "regretless" versioning (+) automatic refresh of dependent data and outputs
R2	(+) manual updates (+) simple scripting (+) plots (-) no support for workflows and more expressive scripting	(+) support for programming language and limited support for multi-modality through custom kernels or magics (-) no spreadsheet data manipulation [1]	(+) multi-modal: programming languages (Python, Scala), query languages (SQL), plots, spreadsheets (+) interoperability through dataset API exposed to all languages
R3	(+) good support for fixing errors manually (-) no automated error detection (-) automated fixes hard to implemented	(-) no convenient support for fixing errors manually (+) powerful programming languages for fixing errors automatically (-) no support for automatically flagging errors and no interface for showing such errors	(+) good support for fixing errors manually using spreadsheets (+) powerful programming languages for fixing errors automatically (+) Lenses [6] for automatic error detection and temporary fixes (+) error reports including automatically generated and manually generated annotations
R4	(-) does not scale[2]	(±) scales using the right extensions to access Spark (-) does not scale to complex pipelines because of lack of automated dependency tracking and automatic refresh	(+) scalable by using Spark as a backend (+) scales to complex pipelines because of versioning, automated refresh and automated dependency tracking
R5	(-) no support for provenance	(-) no support for provenance	(+) workflow provenance maintaining all versions of a notebook (+) flagged errors are maintained automatically (+) sources of uncertainty are tracked by the system [6]
R6	(-) no collaboration support	(±) no collaboration support, users commonly rely on version control systems such as git for collaboration	(+) link sharing for any version of a dataset, plot, or notebook (+) automatic update of dependent datasets and cells

Table 1: Advantages and Limitations of Notebooks and Spreadsheets wrt. the Requirements for Data-centric Work

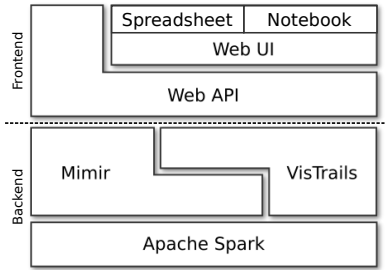

Figure 2: Overview of Vizier's architecture

well as an assortment of related metadata. The Vizier API is built on a backend infrastructure composed of three components: VisTrails, Mimir, and Spark. Workflows are managed by a fork of VisTrails [3], a workflow-based data exploration system that has comprehensive support for both data and workflow-evolution provenance. A unique feature of our VisTrails fork is that all data flow between workflow steps is through Spark dataframes. Where possible, workflow steps are translated directly to spark operators and executed natively in Spark. Mimir [6] implements Lenses, special workflow steps that are the main enabler of data validation and debugging in Vizier. Mimir also implements a limited form of fine-grained provenance management for tracking errors (and other annotations) through workflows.

2.1 A Data-Centric Notebook

In classical "script-centric" notebooks, the fundamental unit of information flow is a kernel: the state of an interactive script interpreter (e.g., IPython). In a data-centric notebook like Vizier, the fundamental unit of information flow is the dataset, i.e., a dataframe or relational table. All cells can access, modify, and produce *datasets* through language-specific APIs. For example, a SQL cell can access datasets as tables in the `FROM` clause and the result of the query can be made available as a dataset for other cells to use. We have found that this more data-centric abstraction takes little away from users, as dataframes are already the lingua franca of many data manipulation toolkits like Pandas. In exchange, Vizier allows native interoperability between multiple languages (presently SQL, Python, and Scala) though this abstraction, and is able to provide the new capabilities described below.

Multi-Modal Interface. Rather than prescribing what method to use to deal with data, Vizier combines a plethora of well-established interfaces as shown in Figure 1. Cells in Vizier notebooks can contain SQL code, Python or Scala code. However, by being data-centric, Vizier also provides specialized cells for higher-level tasks like plotting data or repairing specific classes of data errors semi-automatically [6]. Furthermore, each dataset can be edited through a spreadsheet-like editor view. Changes in the spreadsheets are incorporated as scripts into the notebooks using a spreadsheet update language called Vizual [4] we have developed.

Provenance-Awareness. With individual values in datasets as an atomic unit of information, Vizier is able to capture fine-grained provenance through SQL queries using instrumentation techniques [1]. For example, we can identify which cells (potentially) contribute to one cell of a dataframe, an extremely useful feature for data debugging. By leveraging the provenance graph, Vizier allows cells, rows, columns, or entire datasets to be annotated with documentation that persists through data transformations. Annotated values are highlighted, and Vizier can also generate a convenient listing with all of the documentation for a dataset.

A particularly useful application of this feature are semi-automated data cleaning operations called Lenses [6]. Vizier supports Lenses for schema matching, key repair, type inference, missing value imputation, constraint checking, geocoding, and more. Lens outputs are heuristic, so Vizier documents the choices it makes, allowing users to review and revisit them in the future if they turn out to be problematic.

2.2 Additional Design Choices

We now discuss several additional design choices.

Notebooks are actually Workflows. In contrast to Jupyter, each cell in a Vizier notebook corresponds to a step in a workflow that is transparently maintained by the system using a fork of the VisTrails workflow system [4]. This has the important advantage that changes to a cell are automatically propagated to other cells that depend on the output of the cell. Thus, Vizier is beneficial for complex operations that consist of multiple steps where tracking dependencies among cells and data manually is time consuming (or infeasible), as well as for interactive data curation and exploration, where it is typical for a user to iteratively refine and revise intermediate operations in a workflow.

Scalability and Compatibility through Spark. Vizier relies on Spark to be able to ingest data from a wide variety of sources and for scalable operations over data of any size.

Data Validation. Vizier's Lenses can be used for data validation tasks like enforcing domain, consistency, and key constraints. Violations are reported in the error summary for a dataset, but will not halt the workflow since Lenses make best-effort attempts to repair errors.

Versioning. Since notebooks and spreadsheet operations are internally represented as a workflow based on VisTrails [3], provenance is available for any modification to a workflow and the full (branching) version history of the workflow is tracked. Users can create branches just like in version management systems like git. However, unlike git where the creation of a new version is triggered manual, in Vizier, every change made through any of the user interfaces result in the creation of a new version. Versions are stored compactly recording the changes to the workflow rather than how they differ in terms of datasets.

Collaboration Features. Any version of a notebook, dataset, or plot can be shared through a link produced by Vizier. Furthermore, users can add annotations to spreadsheet cells and these annotations are propagated through operations, queries and lenses. Both user-defined annotations and data errors flagged by lenses are available to all users of a project.

3 DEMONSTRATION DESCRIPTION

Vizier is a fully functional system. The deployment we will be using for the demonstration will contain several preloaded projects, but participants will be allowed to modify these projects and/or create new projects based on other datasets according to their preferences. We will provide a list of links to open data portals to make it easier for participants to find datasets. To not overwhelm participants with too many options initially, we will use a project for the New York cause of death dataset (https://data.cityofnewyork.us/Health/New-York-City-Leading-Causes-of-Death/jb7j-dtam), shown in the accompanying video (https://vizierdb.info/#video_tour), to demonstrate Vizier. Multiple versions of this dataset are available through New York's open data portal. We will use this scenario to illustrate the problems that can arise if the formats and semantics of columns of a new version differ from previous versions and a workflow designed for a previous version is applied to a new version. This dataset is a good example, because different versions of the dataset available through the portal do not use the same assumptions for encoding information. For instance, in one version genders are encoded through single characters (`M` and `F`) while in another version which contains additional new data where gender is encoded as strings (`male` and `female`). We will illustrate that 1) Vizier detects the inconsistencies between the old and the new dataset version, 2) Vizier can pinpoint places in the data that cause the errors, and 3) Vizier provides the necessary tools to fix such problems in the data.

4 CONCLUSIONS

We present *Vizier*, a novel system for multi-modal data curation and exploration that stands out through its support for notebooks and spreadsheets, extensive support for workflow and fine-grained provenance, build-in data curation operations, uncertainty and annotation management features, and its comprehensive collaboration and version control support. Vizier is based on years of research on provenance [1, 3], uncertain data management [6], workflow systems [3], and data curation [6]. Vizier is an open-source system already released at https://github.com/VizierDB. Links to additional resources can be found here: http://www.vizierdb.info/.

ACKNOWLEDGMENTS

This work is supported in part by NSF Awards OAC-1640864 and CNS-1229185, the NYU Moore Sloan Data Science Environment, and DARPA D3M.

REFERENCES

[1] B. S. Arab, S. Feng, B. Glavic, S. Lee, X. Niu, and Q. Zeng. 2018. GProM - A Swiss Army Knife for Your Provenance Needs. *IEEE Data Eng. Bull.* 41, 1 (2018), 51–62.

[2] M. Bendre, V. Venkataraman, X. Zhou, K. Chang, and A. G. Parameswaran. 2018. Towards a Holistic Integration of Spreadsheets with Databases: A Scalable Storage Engine for Presentational Data Management. In *ICDE*.

[3] S. P. Callahan, J. Freire, E. Santos, C. Eduardo Scheidegger, Cláudio T. Silva, and Huy T. Vo. 2006. VisTrails: visualization meets data management. In *SIGMOD*.

[4] J. Freire, B. Glavic, O. Kennedy, and H. Mueller. 2016. The exception that improves the rule. In *HILDA*.

[5] D. Koop and J. Patel. 2017. Dataflow Notebooks: Encoding and Tracking Dependencies of Cells. In *TaPP*.
[6] Y. Yang, N. Meneghetti, R. Fehling, Z. Hua Liu, and O. Kennedy. 2015. Lenses: An On-Demand Approach to ETL. *PVLDB* 8, 12 (2015), 1578–1589.