Outcome-Preserving Input Reduction for Scientific Data Analysis Workflows

Anh Duc Vu
Humboldt-Universität zu Berlin
Berlin, Germany

Timo Kehrer
University of Bern
Bern, Switzerland

Christos Tsigkanos
University of Bern
Bern, Switzerland

ABSTRACT

Analysis of data is the foundation of multiple scientific disciplines, manifesting in complex and diverse scientific data analysis workflows often involving exploratory analyses. Such analyses represent a particular case for traditional data engineering workflows, as results may be hard to interpret and judge whether they are correct or not, and where experimentation is a central theme. Oftentimes, there are certain aspects of a result which are suspicious and which should be further investigated to increase the trustworthiness of the workflow's outcome. To this end, we advocate a semi-automated approach to reducing a workflow's input data while preserving a specified outcome of interest, facilitating irregularity localization by narrowing down the search space for spotting corrupted input data or wrong assumptions made about it. We outline our vision on building engineering support for outcome-preserving input reduction within data analysis workflows, and report on preliminary results obtained from applying an early research prototype on a computational notebook taken from an online community of data scientists and machine learning practitioners.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**.

KEYWORDS

data analysis workflows, automated debugging, data science, notebooks, causality

ACM Reference Format:
Anh Duc Vu, Timo Kehrer, and Christos Tsigkanos. 2022. Outcome-Preserving Input Reduction for Scientific Data Analysis Workflows. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3551349.3559558

1 INTRODUCTION

Essentially all scientific disciplines are generating an ever-increasing amount of data [12]. To derive scientific discoveries, these data sets are analyzed by complex data analysis workflows, employed to analyze, manipulate and investigate data sets in order to apply statistical techniques, spot anomalies, test hypotheses, or check assumptions [20]. Technically, data analysis workflows are series of discrete analysis programs arranged in (often non-linear) pipelines that include facilities to perform data integration, normalization, and filtering, often involving cloud workloads [28]. However, data analysis workflows not only deal with chaining different tools that implement these functionalities, but are a central instrument within the scientific discovery process, which often involves exploratory analysis and experimentation.

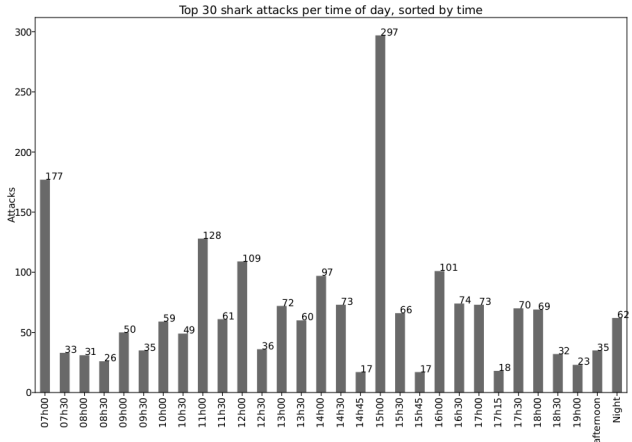

Figure 1: Motivating example: Results from a data analysis workflow over a dataset comprising shark attacks worldwide in the last 100 years.

The trustworthiness of the results of data analysis workflows is critical since, eventually, computational scientists rely on their outcomes for building and validating theories [26]. In the worst case, validity may be jeopardized if the data analysis produces incorrect results [22]. However, exploratory data analyses in scientific computing often yield experimental results which are difficult to interpret and for which it is hard to judge whether they are correct or not [7, 32], which distinguishes them from traditional data engineering pipelines and other kinds of software. Exploratory data analysis in practice is exacerbated further when data sets involved have big data characteristics – when they are large (volume), heterogeneous (variety), change over time (velocity), or are of questionable and varying quality (veracity) [34].

Consider a data analysis workflow over a dataset comprising shark attacks worldwide in the last 100 years, taken from Kaggle[1]. Here, the analysis task is to figure out the number of shark attacks reported per time of the day, and the workflow solving this task is implemented as a computational notebook. Figure 1 shows a plot of the aggregated attacks over certain timeslots. Notably, for 07h00 the number of attacks appears to be quite high in contrast to other times of frequent attacks which are usually around noon and

[1] kaggle.com/mysarahmadbhat/shark-attacks

ASE '22, October 10–14, 2022, Rochester, MI, USA

ACM ISBN 978-1-4503-9475-8/22/10.
https://doi.org/10.1145/3551349.3559558

afternoon. It may certainly be the case that the data indeed show a high amount of attacks for that time; however, there may also be an error in the analysis performed to produce this result, or the data may be corrupted. More generally, assumptions made about the input data may be incorrect and need to be refined [9]. Either way, there are certain aspects of the results which are *suspicious* and which should be further investigated to increase the trustworthiness of the workflow's outcome.

In software engineering, the investigation of suspicious program behavior is generally referred to as *debugging*, which has been tackled from many viewpoints for decades. Most often, assisting techniques such as automated fault localization [1, 38] assume that suspicious program behavior manifests in an error in the program's source code that can be identified by an oracle in the form of a test. The existence of a test suite is therefore necessary to apply these techniques, and their effectiveness depends on the quality of the test suite itself. However, scientific data analyses are explorative, thus making it hard or even impossible to define tests that specify correct behaviour in terms of expected outcomes [15, 16, 25, 30].

On the contrary, the database systems research community has developed techniques for *data provenance* [6], locating the part of input data that is involved in the computation of a result and therefore the cause for it to appear. While this seems to be more adequate to support the investigation of scientific data analysis workflows in the first place, such techniques either require the system to be augmented with additional information gathering capabilities to trace data provenance during execution [14], or they assume knowledge about internals of the workflow semantics [36]. Analogously, the term *explainable AI* has emerged [4], where a research branch pursues explainability by calling for models that are inherently interpretable for humans, such as linear models or decision trees. These techniques require different levels of access to the prediction model, and it remains to be investigated whether they can be adapted for general data processing.

Consequently, we advocate that the investigation of suspicious results of scientific data analysis workflows must accommodate the specific needs of the scientists developing these workflows. Just like the scientific discovery process itself, investigation should be done in an explorative manner. Our motivating example represents a characteristic case where exploratory analysis is required to deduce if the data indeed support shark attacks occurring early in the morning. Intuitively, one would seek to spot the cause of the suspicious outcome while incrementally reducing the data and invoking the workflow. Note that the shark dataset is a CSV file that contains about 25k rows and 24 columns – a rather small dataset compared to what is typically processed in data-centric systems, but still overwhelming for humans to look at. In order to assist the user, a smaller dataset would be useful where only the necessary tuples are contained to reproduce the outcome of interest. To systematically support this reduction by assisting techniques, these techniques should not assume any details about the computational infrastructure, but consider the data analysis workflow as a *black box* in order to be applicable.

To this end, we propose a semi-automated process centered around data reduction to assist debugging an outcome. More specifically, we coin the notion of an *outcome-preserving input reduction* which is supposed to reproduce some outcome of interest of a data analysis workflow to help the user reason about the circumstances surrounding this outcome. In lieu of *fault localization*, we refer to this process as *irregularity localization*, since the outcome investigated may or may not be faulty, given the explorative process. Our motivation can be summarized in the following research question:

> "How can we support semi-automated irregularity localization through outcome-preserving input data reduction in data analysis workflows?"

2 STATE OF THE ART

Traditional research on scientific workflows has focused on optimizing for speed, led by the high-performance computing community. However, with data analysis workflows becoming ubiquitous, there is a changing mindset that human productivity arguably still is the most expensive resource [8]. With our goal of assisting developers in systematically investigating suspicious results, our proposed technique is founded on the idea that input data may be reduced in order to localize an irregularity in either the data or its processing. Accordingly, we classify related work into three major research areas, namely (i) program debugging and software fault localization, (ii) data provenance, and the novel field of (iii) explainable AI.

Program debugging and software fault localization. The idea of systematically narrowing down the search space for locating a bug has been extensively researched; from a high-level point of view, our idea is comparable to program slicing [35, 37], a technique to extract only those parts of a program that are relevant for a particular computation. However, the very idea of slicing relies on the assumption that the source code or some other kind of (formal) program specification [24, 27, 29] is accessible, while we consider the workflow as a black box. Spectrum-based fault localization [1] uses a test suite to assign each program element a suspiciousness score of how likely it is responsible for failing test cases by measuring their involvement in failing and passing tests – heavily depending on a sophisticated test suite [11], unlikely to be available for workflows. Likewise, delta debugging [38] automatically reduces an input file such that each element is necessary to produce an error which is typically spotted through a test suite or crash in a program. Hierarchical Delta Debugging [23] applies this on tree structured data (e.g., XML) – Wang et. al. [33] improved the base algorithm by considering past iterations to build a model on the likelihood of elements being necessary to reproduce a failure.

Data provenance. So-called Why-Provenance [6] is popular in the database community to answer why a specific tuple appears in the result. The Why-Provenance of a result tuple is the set of input tuples that are involved in its computation. Ikeda et al. [13] demonstrate how to utilize provenance to debug workflows by enabling forward tracing of input tuples and backward tracing of result tuples. Titian [14] enables the collection of data provenance for dataflow systems by tagging each record with an id and tracking them through the various data operators. However, data analysis workflows, such as the Jupyter notebook of our motivating example, are often heterogeneous collections of different analyses written in different languages, and we cannot exploit the full capabilities of an underlying data flow system such as Apache Spark, Apache Flink, Apache Airflow, or Nextflow[2]. Moreover, Gulzar et. al. [10]

[2] [spark, flink, airflow].apache.org, nextflow.io

also remark that data provenance often returns excessively much data, and in the worst case the whole input. To tackle this, BigSift combines data provenance with delta debugging to find a minimal set of records that lead to a test failure, which again relies on a test suite that cannot be generally assumed to be available.

Explainable AI. Understanding the underlying operation principles in machine learning concerns the field of explainable AI [4]. There are two major angles that deal with helping humans understand the outputs of such systems. One is to use models that are inherently interpretable, such as linear models and decision trees. The other targets post-hoc techniques to interpret more complex models. A central concept is that of measuring feature importance for a prediction. Permutation feature importance [2] and sensitivity analysis [5] determine the impact of each feature on a model's accuracy by shuffling their values to measure the deviation in accuracy. Lundberg et al. [21] proposed to apply the game-theoric concept of shapley values by modeling features as players in a machine learning prediction game and measuring the average contribution to the game's outcome. This requires testing any possible coalition of players, i.e., all subsets of the feature space. Koh [19] used influence functions to approximate the contribution of each training sample on the model accuracy. However, for non-ML tasks where there is not necessarily a differentiable target function with access to gradients, these techniques can not be readily applied.

3 RESEARCH VISION

Figure 2 illustrates a birds-eye view of the proposed approach. As in a typical exploratory process, a scientist initiates a workflow by submitting some input data to the data analysis workflow (marked as (1) in Fig. 2), which is then processed, yielding an output. A suspicious part of the output may be identified by the user; this is specified in a *debugging question* (marked as (2)). The debugging question is used by an `Oracle` to check whether the property of interest specified by the question holds on the output data (marked as (3)), which is typically the case when the workflow operates on the original input dataset (unless the user specifies an inappropriate debugging question). Then, a `Reducer` facility is supposed to operate upon the input data in order to yield a subset of it. The analysis workflow may be triggered again (marked as (4)) with the reduced input, producing a new output. The oracle is employed to deduce if the outcome of interest (illustrated as a puzzle piece in Fig. 2) is still preserved. This process is triggered iteratively, applying the reduction criterion by the reducer in another iteration and evaluated by the oracle, until some fixpoint is reached. The result (at each iteration) is a reduced input, that can aid in answering the debugging question. This way, the user may keep asking questions about various properties in order to obtain an understanding of the circumstances involving the irregularity sought to be investigated. The approach is semi-automatic: although specification of the debugging question is manual, the rest of the steps are automated. Observe that the workflow is a black box; the proposed approach is agnostic of the workflow internals. In the following, we elaborate on the key elements shown in Fig. 2.

Debugging Question. The debugging question reflects the objective that the user seeks to investigate – it captures something suspicious, and refining the user's understanding is the goal of the exploratory process. The debugging question is thus assumed to be specified by the user who, however, may be assisted by recommendations or by using a domain specific language.

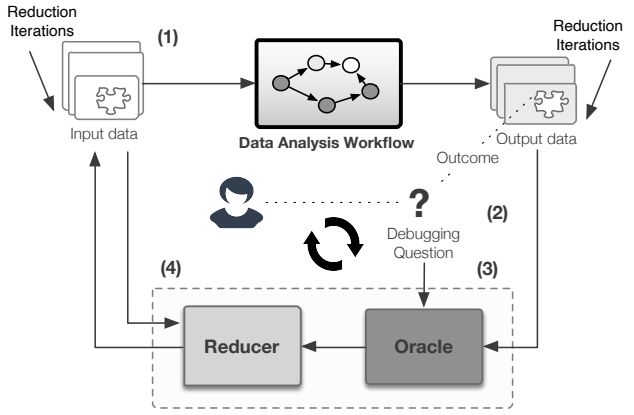

Figure 2: Overview of the approach: General framework, its main components, and developer in the loop.

Reducer. The reducer facility may employ several strategies to reduce the input data. This may occur iteratively, applying and configuring strategies and checking (through the `Oracle`) if the outcome of processing the original input (specified as a debugging question) is preserved over subsequent applications of the workflow. In practice, strategies can employ a variety of techniques, including search enabled by heuristics, machine learning and data mining. In principle, they can be both generic and defined per use case. Explanation techniques for machine learning models may be also incorporated due to presence of AI workloads, or if their assumptions can be relaxed and thus can be applied for general data processing tasks. Note that strategies can also utilize information from previous reduction iterations.

Oracle. The `Oracle` is responsible for checking if the irregularity specified as the debugging question is still preserved in the reduction produced by the application of some strategy. This amounts to evaluating the debugging question as a query.

4 PROTOTYPICAL INSTANTIATION

To investigate the feasibility of the reduction approach advocated for irregularity localization, we realized a proof-of-concept prototypical implementation. We opted for Python as the target language, due to its general popularity, in particular within scientific and data science communities[3]. The shape of the (input and output) data we consider is tabular, in line with our motivating example and as it is customary for many kinds of (statistical) scientific software [31]. Debugging question specification is enabled by Pandas Dataframes[4], implementing both data handling and the oracle.

Our goal is to explore feasibility and assess the design of the system realizing the overall architecture of Fig. 2. Thereupon, we realized two strategies, instantiating the `Reducer` element of Fig. 2 – one based on delta debugging, representing a base case, and a more involved one utilizing behavioral similarity, illustrating the potential that more advanced techniques can have. We choose those

[3]https://spectrum.ieee.org/top-programming-languages-2022
[4]pandas.pydata.org

strategies as representative cases where one reflects a straightforward but generically applicable approach, and the other one exploits characteristics of the underlying data.

Reduction Strategy 1. The *ddmin* algorithm [38] implementing delta debugging is a generic approach which works for arbitrary sets of items. For the tabular shape of data as in our example, there are two dimensions where a reduction can be applied; the table rows and columns. However, note that *ddmin* can only be applied to get rid of irrelevant rows. Columns can not be simply reduced by removing them, as removing columns from a table is a change in its format; which is highly likely to disrupt the data analysis workflow. The strategy we implement therefore randomly mutates values in a column, in effect exploiting randomization to assess if they have an effect on the outcome.

Reduction Strategy 2. Under the assumption that similar data induces similar behaviour, we propose a more advanced strategy combining fault isolation and similarity search. Adopting the fault isolation algorithm from [17], we compute two reduced datasets from the original input that differ in only one element – one dataset reproduces the specified outcome and the other does not. During this process, there should be data that got removed in the process without affecting the outcome. These serve as the basis for similarity search, which exploits the structural nature of the data at hand. If we find data tuples similar to the ones that have been removed without affecting the outcome, it is likely that we can remove them as well. In case that no tuples are removed, this is a strong indicator that the data in the current iteration has reached a minimal point. We define similarity of two records as the average similarity over each column value. For each value, we use classical distance metrics depending on their data type. For our prototypical instantiation, these are the normalized Euclidean distance for numerical values, the Levenshtein distance for strings, and equality for categorical data. Data types of each column are derived automatically through profiling.

Preliminary Results. Back to our shark attack example of Sec. 1, recall that the user is investigating the unusual spike in attacks early in the morning (Fig. 1). The user specifies the debugging question as a Pandas query of "(Time=="07h00" & Count == 177)" to capture the outcome of interest. Figure 3 depicts a profile of the reduced input dataset delivered by our tool. The minimized dataset comprises 177 tuples and one column that are needed to reproduce the behaviour specified by the debugging question. Observe that only column 'Time' has been deemed relevant, and the relevant tuples take the values of 'Morning', 'Evening' and '07h00'. This represents a strong hint to the user; some data transformation applied within the workflow did yield these irregular values.

As is typical in the domain, data cleaning can have unintended effects; in this case, some cleaning was irregularly applied to group the strings 'Morning' and 'Evening' with '07h00'. The user can then examine the relevant part of the workflow (black box of Fig. 2). For the example presented, there exists a function `convert_time_text()` that converts time given as text into clock time that contained the bug. It is a up to the user to assess whether mapping 'Morning' and 'Evening' to '07h00' is intended functionality or not.

In our preliminary evaluation of the shark example, applying the *ddmin* strategy took 3947 iterations to reduce 25723 rows to 177 rows, with a running time of approximately 10 minutes on a Macbook with a 2 GHz Quad-Core Intel Core i5 CPU and 16 GB 3733 MHz LPDDR4X Ram. In comparison, the isolation and similarity strategy took 59 iterations and about 15 seconds for the rows. Both ran 7 iterations to reduce 24 columns to 1 column. While we do by no means argue for the generalizability of our results, for this particular example, the data-aware approach was superior to the naive one, since the analysis workflow was aggregating data tuples with the same values in the relevant columns. As such, compared to randomly guessing a data reduction, the use of similarity could easily guide the reducer to remove irrelevant data. We believe this illustrates the high potential of utilizing information about the data to guide reduction.

Figure 3: Data profile of the reduced input dataset, serving as a starting point to localize the irregularity observed in our motivating example.

5 CHALLENGES AND RESEARCH AGENDA

Exploratory data analysis within scientific computing very often yields results for which it is hard to judge whether they are correct or not. To support irregularity localization in such workflows, we advocate the employment of reduction techniques to produce minimal data sets that preserve a specified outcome, to aid debugging. To realize an end-to-end framework, we identify several open software engineering challenges involved in answering the research question set previously, spanning various levels of abstraction.

Data shapes and efficient strategies. Scientific workflows revolve around data; as such, the shape of the data as well as processing and size characteristics naturally affect the choice and development of reduction strategies. Firstly, assuming a specific structure in input datasets (e.g., tabular, hierarchical, graph-like) can give rise to specialized strategies and heuristics for reduction. As such, construction and development of a portfolio of strategies able to handle different data shapes efficiently is a priority. Secondly, data-heavy workflows often process huge amounts of data and can therefore have rather long running times (and thus cost), which affects the number of feasible iterations that can be performed for reduction. It is thus essential to evaluate strategies against (i) *efficiency*, defined as the inverse of the cost of iterations needed, and (ii) *effectiveness*, in terms of the size of the reduction they achieve. Finally, trade-offs between such efficiency and effectiveness for these strategies should be gauged, so that a designer can do appropriate dimensioning.

Effective specification of debugging questions. An important step is the specification of debugging questions; developers should be supported effectively in their formulation. Although, in general, a user may utilize the full expressiveness of their programming language of choice, specialized methods can aid in specification. A natural direction is interactive question formulation, such as within computational notebooks [18]. Due to the exploratory nature of the process at hand, there is minimal knowledge about what results should be; as such, recommendation facilities aiding specification of potentially questionable outcomes appear promising. A starting point can be to recommend debugging questions targeting

outliers. DSLs and similar techniques can be further integrated, incorporating features such as outlier detection, and in tandem with visualization techniques for incremental specification [3].

End-user validation. An empirical investigation should be performed to assess whether the approach helps users in debugging and understanding their workflows *end-to-end*. In particular, we treat as foundational an investigation of effectiveness over selected workflows that pose challenges for analysis, including remote sensing, biomedical image analysis and space downstream data processing. User studies within such endeavors should determine how useful reductions are in helping users spot irregularities in their data.

ACKNOWLEDGMENTS

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) Project-ID 414984028 – SFB 1404 FONDA.

REFERENCES

[1] Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. 2007. On the accuracy of spectrum-based fault localization. In *Testing: Academic and industrial conference practice and research techniques-MUTATION (TAICPART-MUTATION 2007)*. IEEE, 89–98.

[2] André Altmann, Laura Toloşi, Oliver Sander, and Thomas Lengauer. 2010. Permutation importance: a corrected feature importance measure. *Bioinformatics* 26, 10 (2010), 1340–1347.

[3] Alessio Arleo, Johannes Sorger, Christos Tsigkanos, Chao Jia, Roger A. Leite, Ilir Murturi, Manfred Klaffenböck, Schahram Dustdar, Michael Wimmer, and Silvia Miksch. 2019. Sabrina: Modeling and Visualization of Financial Data over Time with Incremental Domain Knowledge. In *30th IEEE Visualization Conference, IEEE VIS 2019 - Short Papers, Vancouver, BC, Canada, October 20-25, 2019*. IEEE, 51–55. https://doi.org/10.1109/VISUAL.2019.8933598

[4] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. 2020. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information fusion* 58 (2020), 82–115.

[5] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. 2010. How to explain individual classification decisions. *The Journal of Machine Learning Research* 11 (2010), 1803–1831.

[6] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. 2001. Why and where: A characterization of data provenance. In *International conference on database theory*. Springer, 316–330.

[7] Jeffrey C Carver, Richard P Kendall, Susan E Squires, and Douglass E Post. 2007. Software development environments for scientific and engineering software: A series of case studies. In *29th International Conference on Software Engineering (ICSE'07)*. Ieee, 550–559.

[8] Ewa Deelman, Tom Peterka, Ilkay Altintas, Christopher D Carothers, Kerstin Kleese van Dam, Kenneth Moreland, Manish Parashar, Lavanya Ramakrishnan, Michela Taufer, and Jeffrey Vetter. 2018. The future of scientific workflows. *The International Journal of High Performance Computing Applications* 32, 1 (2018), 159–175.

[9] Sainyam Galhotra, Anna Fariha, Raoni Lourenço, Juliana Freire, Alexandra Meliou, and Divesh Srivastava. 2021. DataExposer: Exposing Disconnect between Data and Systems. *arXiv preprint arXiv:2105.06058* (2021).

[10] Muhammad Ali Gulzar, Matteo Interlandi, Xueyuan Han, Mingda Li, Tyson Condie, and Miryung Kim. 2017. Automated debugging in data-intensive scalable computing. In *Proceedings of the 2017 Symposium on Cloud Computing*. 520–534.

[11] Simon Heiden, Lars Grunske, Timo Kehrer, Fabian Keller, Andre Van Hoorn, Antonio Filieri, and David Lo. 2019. An evaluation of pure spectrum-based fault localization techniques for large-scale software systems. *Software: Practice and Experience* 49, 8 (2019), 1197–1224.

[12] Anthony JG Hey, Stewart Tansley, Kristin Michele Tolle, et al. 2009. *The fourth paradigm: data-intensive scientific discovery*. Vol. 1. Microsoft research Redmond, WA.

[13] Robert Ikeda, Junsang Cho, Charlie Fang, Semih Salihoglu, Satoshi Torikai, and Jennifer Widom. 2012. Provenance-based debugging and drill-down in data-oriented workflows. In *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 1249–1252.

[14] Matteo Interlandi, Kshitij Shah, Sai Deep Tetali, Muhammad Ali Gulzar, Seunghyun Yoo, Miryung Kim, Todd Millstein, and Tyson Condie. 2015. Titian: Data provenance support in spark. In *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, Vol. 9. NIH Public Access, 216.

[15] Upulee Kanewala and James M Bieman. 2014. Testing scientific software: A systematic literature review. *Information and software technology* 56, 10 (2014), 1219–1232.

[16] Diane Kelly and Rebecca Sanders. 2008. The challenge of testing scientific software. In *Proceedings of the 3rd annual conference of the Association for Software Testing (CAST 2008: Beyond the Boundaries)*. Citeseer, 30–36.

[17] Lukas Kirschner, Ezekiel Soremekun, and Andreas Zeller. 2020. Debugging inputs. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 75–86.

[18] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. 2016. *Jupyter Notebooks-a publishing format for reproducible computational workflows*. Vol. 2016.

[19] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International conference on machine learning*. PMLR, 1885–1894.

[20] Ulf Leser, Marcus Hilbrich, Claudia Draxl, Peter Eisert, Lars Grunske, Patrick Hostert, Dagmar Kainmüller, Odej Kao, Birte Kehr, Timo Kehrer, Christoph Koch, Volker Markl, Henning Meyerhenke, Tilmann Rabl, Alexander Reinefeld, Knut Reinert, Kerstin Ritter, Björn Scheuermann, Florian Schintke, Nicole Schweikardt, and Matthias Weidlich. 2021. The Collaborative Research Center FONDA. *Datenbank-Spektrum* (2021).

[21] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. *Advances in neural information processing systems* 30 (2017).

[22] Greg Miller. 2006. A scientist's nightmare: software problem leads to five retractions.

[23] Ghassan Misherghi and Zhendong Su. 2006. HDD: hierarchical delta debugging. In *Proceedings of the 28th international conference on Software engineering*. 142–151.

[24] Christopher Pietsch, Manuel Ohrndorf, Udo Kelter, and Timo Kehrer. 2017. Incrementally slicing editable submodels. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 913–918.

[25] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2019. A large-scale study about quality and reproducibility of jupyter notebooks. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 507–517.

[26] Rebecca Sanders and Diane Kelly. 2008. Dealing with risk in scientific software development. *IEEE software* 25, 4 (2008), 21–28.

[27] Gabriele Taentzer, Timo Kehrer, Christopher Pietsch, and Udo Kelter. 2018. A formal framework for incremental model slicing. In *International Conference on Fundamental Approaches to Software Engineering*. Springer, 3–20.

[28] Christos Tsigkanos and Timo Kehrer. 2016. On Formalizing and Identifying Patterns in Cloud Workload Specifications. In *13th Working IEEE/IFIP Conference on Software Architecture, WICSA 2016, Venice, Italy, April 5-8, 2016*. IEEE Computer Society, 262–267.

[29] Christos Tsigkanos, Nianyu Li, Zhi Jin, Zhenjiang Hu, and Carlo Ghezzi. 2020. Scalable Multiple-View Analysis of Reactive Systems via Bidirectional Model Transformations. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*. IEEE, 993–1003. https://doi.org/10.1145/3324884.3416579

[30] Melina Vidoni. 2021. Evaluating unit testing practices in r packages. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1523–1534.

[31] Melina C Vidoni. 2021. Software Engineering and R Programming: A Call for Research. *R J.* 13, 2 (2021), 600.

[32] Thomas Vogel, Stephan Druskat, Markus Scheidgen, Claudia Draxl, and Lars Grunske. 2019. Challenges for verifying and validating scientific software in computational materials science. In *2019 IEEE/ACM 14th International Workshop on Software Engineering for Science (SE4Science)*. IEEE, 25–32.

[33] Guancheng Wang, Ruobing Shen, Junjie Chen, Yingfei Xiong, and Lu Zhang. 2021. Probabilistic Delta debugging. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 881–892.

[34] Jonathan Stuart Ward and Adam Barker. 2013. Undefined by data: a survey of big data definitions. *arXiv preprint arXiv:1309.5821* (2013).

[35] Mark Weiser. 1984. Program slicing. *IEEE Transactions on software engineering* 4 (1984), 352–357.

[36] Weiyuan Wu, Lampros Flokas, Eugene Wu, and Jiannan Wang. 2020. Complaint-driven training data debugging for query 2.0. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1317–1334.

[37] Baowen Xu, Ju Qian, Xiaofang Zhang, Zhongqiang Wu, and Lin Chen. 2005. A brief survey of program slicing. *ACM SIGSOFT Software Engineering Notes* 30, 2 (2005), 1–36.

[38] Andreas Zeller and Ralf Hildebrandt. 2002. Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering* 28, 2 (2002), 183–200.