The Journal of Systems & Software 197 (2023) 111573

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

In Practice

Automated data validation: An industrial experience report☆

Lei Zhang [a,*], Sean Howard [b], Tom Montpool [b], Jessica Moore [b], Krittika Mahajan [b], Andriy Miranskyy [a,*]

[a] Department of Computer Science, Ryerson University, Toronto, Canada
[b] Environics Analytics, Toronto, Canada

ARTICLE INFO

Article history:
Received 20 September 2021
Received in revised form 3 August 2022
Accepted 24 November 2022
Available online 28 November 2022

Keywords:
Data validation
Test automation
Data science
Software engineering best practice

ABSTRACT

There has been a massive explosion of data generated by customers and retained by companies in the last decade. However, there is a significant mismatch between the increasing volume of data and the lack of automation methods and tools. The lack of best practices in data science programming may lead to software quality degradation, release schedule slippage, and budget overruns. To mitigate these concerns, we would like to bring software engineering best practices into data science. Specifically, we focus on automated data validation in the data preparation phase of the software development life cycle.

This paper studies a real-world industrial case and applies software engineering best practices to develop an automated test harness called RESTORE. We release RESTORE as an open-source R package. Our experience report, done on the geodemographic data, shows that RESTORE enables efficient and effective detection of errors injected during the data preparation phase. RESTORE also significantly reduced the cost of testing. We hope that the community benefits from the open-source project and the practical advice based on our experience.

1. Introduction

Big data and data analytics are revolutionizing modern business processes of sales and marketing. McKinsey Analytics's report shows that big data and data analytics technologies have served as a fundamental technology for artificial intelligence and also enabled a new big data ecosystem (McKinsey Analytics, 2018). IDC says that worldwide data volume will grow 61% to 175 ZB by 2025 (IDC, 2018). Wikibon predicts that the big data revenues for software and services will reach US $103 B in 2027 with an annual growth rate of 10.48% (Columbus, 2018).

Besides business opportunities, big data also brings challenges to data scientists. Poor data quality leads to failures of their machine learning models. How to improve the data quality is an active topic in both academia and industry. Data validation is known as an essential process to ensure data quality so that the data are both correct and meaningful for the next step of data analytics. However, due to the large volume, the fast velocity, and the wide variety of generated big data, the data science community is seeking for a more efficient way to ensure data quality. Modern data validation tools have the ability to automate this process to improve the efficiency (Bonter and Cooper, 2012; Paygude and Devale, 2013a,b; Rathika and Arcokiam, 2014; Hynes et al., 2017; Polyzotis et al., 2017; Breck et al., 2019). Automation is also one of the most important concepts in continuous integration. However, to the best of our knowledge, automated testing methods and tools are still lacking a mechanism to detect data errors in the datasets, which are updated periodically, by comparing different versions of datasets.

There is another underlying challenge. Data science programmers implementing data analysis algorithms often may not have a computer science or software engineering training as they come from other branches of science, such as mathematics and statistics. Thus, many data science programmers will not have formal training in developing and maintaining complex software solutions. Based on the literature (Cao, 2017; Zhang et al., 2017) and discussions with practitioners, the programming practices of data science programmers often resemble those of programmers from 1960s that triggered a software crisis leading to the creation of software engineering in 1968 (Wirth, 2008). Similar issue of lacking best practices also happens in scientific-application software (Kelly, 2007). These issues are currently prevalent in the data science community, where existing solutions are of low quality, unmaintainable, and non-evolvable (Kim et al., 2016, 2018). All of this leads to significant economic loss, as the cost of evolving data science products is higher than it should be.

☆ Editor: Marcos Kalinowski.

* Corresponding authors.
E-mail addresses: leizhang@ryerson.ca (L. Zhang), avm@ryerson.ca (A. Miranskyy).

https://doi.org/10.1016/j.jss.2022.111573
0164-1212/

Theoretically, one can try to address these issues by forcing existing rigorous software engineering best practices on data science programmers; realistically, it will (almost surely) work ineffectively (Kim et al., 2016, 2018). This will happen because the programmers lack foundational training and may not have time, interest, or energy to learn laborious practices. Thus, one needs to either tailor existing software engineering techniques or create new lightweight software engineering techniques that would be adopted by data science programmers. Our ultimate **goal** is to improve the state-of-the-practice and to bring lightweight yet rigorous software engineering into data science.

Automation is crucial for efficient delivery in data-driven software development (Amershi et al., 2019), especially in two areas: (1) data preparation and (2) deployment. In this paper, we focus on the former — creating automated data validation for verifying the correctness of data preparation and cleansing processes.

In a nutshell, our **contributions** in this paper are two-fold. First, we report issues that a team of data scientists experience while processing geodemographic[1] datasets. The geodemographic data have to be updated periodically (e.g., due to arrival of new census data). Such an update entails data preparation and cleansing, which is often error-prone.

Second, to combat the issues experienced during the data update, we present an automated data validation framework called RESTORE. This framework automates the statistical tests introduced by the team as part of the generation of best practices. RESTORE compares the previous version/vintage[2] of a dataset with a new one and reports potential issues. The framework is written in R language (R. Core Team, 2017) and is released as an open-source R package on GitHub (Zhang et al., 2021).[3]

By adopting the RESTORE R package in an industrial setting, we seek an answer to the question — **does the RESTORE package improve the efficiency of the data validation procedure, i.e., does it help identify data errors in less time and with fewer human resources?**

RESTORE introduces a set of test cases that our team found useful empirically. However, this set is not exhaustive. Thus, we designed RESTORE so that it can be easily extended with additional test cases, if other data science teams decide to introduce new test cases that would help them in validating their data. We welcome contributions to the code via GitHub. Moreover, the team come up with a set of best practices based on our experiences, and we hope data science practitioners who are facing data validation challenges benefit from our take-away messages.

The rest of the paper is structured as follows. In Section 2, we introduce the background of data validation, geodemographic data, and our industrial settings. Section 3 discusses our proposed method for automated data validation. Section 4 presents the details of data tests. In Section 5, we analyze the pros, cons, and assumptions of our method. Section 6 introduces the interface of RESTORE, our validation, threats to validity, potential extensions of RESTORE, and take-away messages. Section 7 depicts related work. Finally, Section 8 concludes the paper.

2. Background

In this section, we first discuss the importance of data validation (Section 2.1). Then, we review the existing solutions (Section 2.2). We also give an introduction of the characteristics of geodemographic data (Section 2.3). Finally, we illustrate the data development process in our industrial settings (Section 2.4).

2.1. Importance of data validation

Data-driven software development consists of both data-oriented stages (e.g., data collection and data cleaning) and model-oriented stages (e.g., model training and model evaluation) (Amershi et al., 2019). The first stage is often data collection where data scientists look for and integrate available datasets or collect their own.

A typical data analytics company[4] will have to collect multiple datasets at this stage, many of which will be updated regularly. For example, a ledger of trades from a foreign office of a bank may be updated daily, macroeconomic metrics data — quarterly, and census data — yearly. During an update, values of variables are typically refreshed. In addition, observations and variables can be added or removed in the new version of the dataset. Once updated data are received, the data development team will clean and transform the data. Then the data will be passed to software development and testing teams that will ingest the data into data science software products that customers will use to access and analyze the data.

The scenario above is a typical Extract, Transform, and Load (ETL) process. Our proposed data validation technique is feasible in any cases where ETL is required, such as machine learning or business analysis. The sequence may vary on a case-by-case basis, but the abstract principle remains the same. Fig. 1 shows how data validation techniques can be incorporated into the ETL pipeline.

In ETL processes, data transformation routines (manual and automatic) may be defective, leading to errors in the resulting output. Then, the defective output will be used as input data for analytical models, leading to errors in data analytics, especially in machine learning models where input data errors can be amplified over a feedback loop. Such triggers may lead to various failures on the software side, ranging from a failed query in the analytics system (which is easy to detect) to incorrect results (which may be harder to expose). For example, suppose a report suggests that the overall population of Canada is 100 million. In that case, it is an apparent data defect that is easy to spot because the correct answer (as per the 2016 Census) is 36.29 million. However, suppose the report tells us that the population of Canada is 37 million. In that case, a tester may need to verify the numbers manually and must have some context to assess the data appropriately.

As discussed, data validation is the last step of data development before loading the data to the target platform, which is a counterpart of validation testing in software testing. For example, bugs of internal scripts used for ETL can generate data defects. However, such data defects cannot be revealed by testing the software code but can be reported by validating the data. Below, we list some of the root causes that can lead the input data to incorrectness.

1. The raw data may be corrupted, e.g., due to incorrect extraction from an external source.
2. Data scientists may perform some data transformations manually, which injects some errors.
3. The ETL scripts may contain hard-coded values, which worked for the previous version of the dataset but not for the new version. To better illustrate this case, suppose that we have an "if" block, i.e., "if $x \neq 10$", which works for values <10, but the same "if" block can trigger an error in the new version where $x = 11$.

[1] Geodemography is an area of market research, specializing in profiling economic and demographic characteristics of geographical areas (Goss, 1995). Geodemographic datasets are typically hierarchical, i.e., tree-like. For example, a group of postal codes belong to a town, the group of towns belong to a county, and a group of counties belong to a province.

[2] From hereon, we will use the term *version* and *vintage* interchangeably.

[3] https://github.com/miranska/restore.

[4] Data analytics companies provide data analytics services by analyzing the acquired data to assist businesses such as software development, market analysis, improving operational efficiency, etc.

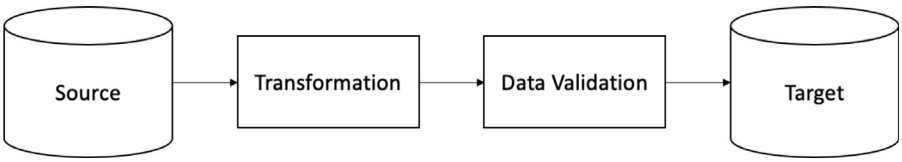


Fig. 1. Data validation in ETL pipeline.

Here, we pick two concrete examples from our demos in RESTORE, which can be found in the file of `analysis_results_hierarchy.xlsx` (Zhang et al., 2021), to illustrate what data defects are and how we can detect them. In the first example, we show a synthetic case where the number of observations changes significantly from the old vintage of the dataset to the new vintage of the dataset (i.e., 16,085 versus 16,018 is reasonable rather than 16,085 versus 160, which is differed by two orders of magnitude). Such a data defect can be detected by magnitude test, i.e., any differences greater than one order of magnitude should be automatically detected and reported to data scientists for further investigation. In the second example, to assess monotonic relationships, we perform the Spearman rank-order correlation test on two variables, and we set a threshold, which is 0.8. As a result, any correlation coefficient with a value of less than 0.8 will be added to the "alert" list and manually evaluated by data scientists.

If no anomalies are revealed during the data preparation stage, these anomalies may pass on to the software developers or testers, leading to failures in system tests. The defective data will be returned to the data development team for a fix. The data development team will now have to fix a defect, then test and reload the data. These delays lead to schedule slippage and budget overruns. Therefore, we need to find a solution to identify such defects as early as possible, saving time and money. In this paper, our goal is to detect data-related defects automatically, streamline data update schedules, and reduce the cost of detecting and fixing the defects.

2.2. Existing solutions

There are many test frameworks for testing database engines and business logic that alters the data in the databases (Kapfhammer and Soffa, 2003; Maule et al., 2008; Haraty et al., 2002; Nanda et al., 2011; Haftmann et al., 2005). Some automated data validation methods for data migration have been proposed (Paygude and Devale, 2013a,b; Rathika and Arcokiam, 2014). In addition, some automated database testing frameworks (DbFit Community, 2019; DbUnit Community, 2019; NDbUnit Community, 2019; DBTD Project, 2019; Microsoft, 2019) are developed to make sure that the previously captured analytics SQL queries execute successfully in the current version. It is important to monitor the quality of data fed to analytics or machine learning models because errors in the input data can nullify the accuracy. Thus, data validation frameworks for data analytics, especially for machine learning models recently, are proposed (Gao et al., 2016; Bonter and Cooper, 2012; Sadiq et al., 2004; Polyzotis et al., 2017; Breck et al., 2019; Hynes et al., 2017).

We are not interested in database testing systems. The problem we aim to solve is complementary. We focus on data validation at an early stage — before a dataset is loaded into the database. More specifically, we are interested in validating a new dataset by comparing it against the predecessor (like regression testing in software engineering). To reach our goal, we need to examine the changes in the data preparation step and flag the erroneous records and variables.

2.3. Geodemographic data

As mentioned in Section 1, geodemographic datasets have two characteristics. First, geodemographic datasets are typically hierarchical, e.g., a group of postal codes belong to a town, a group of towns belong to a county, and so on. Second, geodemographic data are usually updated periodically, e.g., due to arrival of new census data.

As an example of periodically updated datasets, many demographic data and surveys (leveraged in geodemography) are updated annually by national census organizations or primary research companies. These data, in turn, get ingested by companies around the globe to improve business decisions. The data ingestion process leads to dataset refreshment and/or transformation, which may introduce new data flaws (e.g., incorrect values) or defects (e.g., changed/missing attributes) in the software.

The consequences of such errors vary. For example, if the transformed dataset does not have a required variable, the software doing data analysis on this transformed data may fail as it would be unable to find the variable. Such an error would be detected fairly early in the testing of software systems. However, an error may be more subtle: all the variables would be present, but the values of these variables are incorrect, leading to incorrect results generated by the business intelligence software. In one example, a sample report may suggest that the average price of a house for area B is $50K while for area C it is $10M. Both numbers are extreme, but not outside of the realm of possibility. Thus, an analyst may need to manually verify both numbers and must have some context to appropriately assess the resulting numbers. This manual verification is arduous.

Note that our proposed solution — RESTORE — is designed to deal with hierarchical data, but it can be applied to "flat" datasets too by setting hierarchical depth to zero. Therefore, we hope that this framework will be helpful to any data scientist dealing with flat or hierarchical data that requires periodic update.

2.4. Environics analytics

It is important to understand how the data are developed and tested in a typical data analytics/data science project. In this research project, we provide a concrete example to illustrate a typical procedure of data development and testing, based on the process adopted by Environics Analytics, Toronto, Canada (abbreviated to EA in this paper). EA specializes in geodemography and marketing analytics and builds standard and custom data-driven solutions for their clients. Many of their data and services are provided using a Software-as-a-Service (SaaS) approach (Mell and Grance, 2011). These services require datasets to be updated multiple times per year. During the update, anomalies in new vintages of datasets may introduce defects in services. The quality assurance team used to manually test the datasets to detect and eliminate the defects, but this process was time- and human-resource-consuming.

EA's SaaS platform hosts data built and maintained by EA, as well as data supplied by EA's partners or clients. Data supplied by partners or clients can be in a variety of formats. Their most common data structure is a tabular dataset that consists of various levels of geographic information. This data architecture has its

advantages because it allows for the necessary flexibility to work with different types and sizes of data. In this paper, we assume that all datasets are in tabular format (or can be converted to this format).

Before an automated data testing tool is adopted, the dataset development process in EA is as follows.

1. The data team creates a dataset (either a new one or an update of an existing one).
2. This dataset is loaded into a staging database by the data team.
3. The software development and quality assurance/testing teams execute a mixture of automated and manual test workloads (against the application) mimicking customers' behavior (e.g., select a particular geographic area and then run a house prices report). Under the hood, the software layer issues analytic (read-only) queries to the staging database. As part of the software testing, data in analytics reports are assessed, resulting in possible data errors to be uncovered.
4. If failures (such as those discussed in Section 1) in the analytics reports are observed during the execution of the workloads, a bug report is issued for further investigation. Data bugs may exist in various forms: e.g., errors in raw data, errors in the calculation of "constructed" variables as part of the load into the staging database, and errors generated by how the application handles the data.

Once the data team fixes the defects, this team recreates the dataset as necessary, reloads it into the staging database, and hands it over to the software development team for testing (basically, rerunning the above process). This process repeats itself until all the data-related defects are eliminated. Then the dataset is loaded into a production database, and the product is made available to a customer. As mentioned in Section 1, the process is time-consuming and may significantly delay the release (from days to weeks) of the product to customers.

3. Our solution

We develop a theoretical foundation for comparison of various versions of the datasets. Rather than performing the exact comparison, we explore methods needed to compare distributions of the data. For example, we compare distributions of the data in a pair of releases (e.g., using Kolmogorov–Smirnov test Smirnov, 1939 or Mann–Whitney U test Mann and Whitney, 1947). We implement these theoretical findings in an automated regression testing framework. Regression testing is a type of testing, which ensures that the existing functionality of a software product is not broken with new changes, i.e., the functionality does not regress (Huizinga and Kolawa, 2007; Lewis, 2000). In our case, the functionality is data-centric.

The framework enables automatic testing of datasets immediately after generation of a new version of the data, thus, ensuring that defects in the data are captured early in the process, before the dataset is shipped off by the data development team. We do not alter the original inputs (while following best practices of functional programming). Instead, we take a sequence of transformations of datasets and analyze the results of the transformation. The optimal goal of this study is to shorten the development and testing cycle, reducing the probability of schedule slippage and freeing resources to focus on more complex workloads, thus,

1. Improving overall product quality (as teams will have more time and resources to identify complex defects that otherwise would be "masked" by simpler defects, which can be caught by the automated regression testing van Megen and Meyerhoff, 1995; Rafi et al., 2012), and
2. Reducing development costs (the savings will manifest themselves because the cost of creating, maintaining, and executing test cases will be lower than the cost of manual testing) (Dustin et al., 1999).

These steps of automated data regression testing are graphically depicted in Fig. 2, which shows the following process (discussed in details in Sections 4 and 6).

1. Load both old and new vintages of the dataset into RESTORE.
2. Apply a set of test to verify and validate the integrity of the new vintage.
3. Generate the final report which is exported in human- and machine-readable formats.

Note that the report generated by RESTORE needs to be manually reviewed by the data scientists. The data scientists will use their experiences to decide whether a further investigation is necessary. The RESTORE package helps the data scientists to automate mundane data validation tasks and detect potential data defects as soon as possible (in the data preparation stage). We will discuss the assumptions and limitations of the proposed method in Section 5.4. Let us first look at the details of the tests used to compare the vintages.

Table 1
Example of a dataset vintage; $v_{\cdot}$ is a variable name.

Key	Hierarchy level	v_1	...	v_N
1	National	100	...	500
2	City	3	...	16
...	...	...	...	...
M	National	12	...	124

4. Tests' description

To address our problem (i.e., validation of the modified dataset), a set of tests performs an approximate comparison (rather than exact comparison as done by DbFit Community (2019)) of the datasets. Based on practical experiences, we create ten groups of tests, which will be discussed in details below.

We also need to define success criteria, which consists of numeric thresholds, for these tests. The criteria are defined based on our practical experience, where we find that these values provide a large number of true data defects while keeping the number of false defects low. We cannot guarantee that these values are optimal for any dataset; rather, they can be treated as a set of good starting values and adjusted based on a particular use-case and the needs of a data tester.

Below we give details of our tests grouped into three categories: high-level tests dealing with metadata, tests of paired observations, and tests leveraging the results of the paired tests (which we deem higher-order tests). These tests are discussed in Sections 4.1, 4.2, and 4.3, respectively.

4.1. High-level testing of vintages

An example of a dataset vintage with N variables[5] and M observations is given in Table 1. Note that the 'Key' values are not necessarily numeric. The only constraint is that the 2-tuple of 'Key' and 'Hierarchy Level' should be unique for every row (i.e., observation).

We now perform three groups of high-level tests assessing the characteristics of the vintages as follows: 1. comparing attributes of the variables, 2. checking the variables for missing observations, and 3. counting discarded observations.

[5] The term variable is synonymous to column or feature, depending on the reader's background.

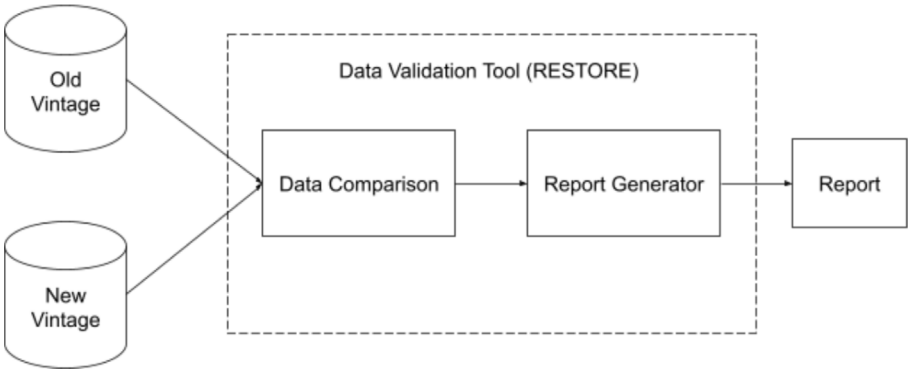


Fig. 2. Comparison of old and new vintages of the dataset.

4.1.1. Variables' attributes comparison

Rationale: We perform a set of "sanity-check" tests, comparing high-level characteristics of the datasets (i.e., metadata), such as the number of rows and columns. If the numbers do not match, this may be a cause for concern.[6] Here, we assume that the datasets have no significant changes. Details of our assumptions can be found in Section 5.4.

Method: We obtain three items for each of the vintages, old and new: the number of variables, the names of variables, and the number of observations in the dataset. We then compare these items.

Success criteria: If the number of variables, the names of the variables, and the number of observations are the same, then the test passes. If the number of variables or the number of observations is not identical between the old vintage and the new vintage of a dataset, the test fails and this mismatch gets reported. If a variable, present in the old vintage, is not present in the new vintage (or vice versa), then it is also considered a failure and the name of the variable is added to the report.

4.1.2. Missing (NA) observations

Rationale: Typically, a clean dataset should not have missing observations in a variable.

Method: Thus, for each 'Variable Name' (e.g., for each v_i in $v_1, \ldots, v_N$ in Table 1) and 'Hierarchy Level', we search for missing observations (in R such observations are marked as NA). This process is done individually for old and new vintages of the dataset.

Success criteria: A 'Variable Name' and 'Hierarchy Level' pair that has zero missing observations passes the test; otherwise, it gets reported.

4.1.3. Discarded observation count

Now we can join the old and new vintages of the dataset, so that we can perform pairwise tests for each variable (as will be discussed in Section 4.2). We perform the inner join (in the relational algebra sense of the term Maier, 1983) on the 'Key' and 'Hierarchy Level' columns (shown in Table 1) of the old and new vintages, discarding the observations that are present in only one of the vintages. Before moving to the pairwise tests, we will perform one last metadata test, based on the count of discarded observations.

Rationale: In practice, we found out that paired observations are more valuable for detecting defects than the non-paired ones (as they contain more information about changes to the dataset). However, the observations that did not make it into the inner join of the old and new vintage may indicate a defect in the data preparation process.

Method: Count the number of observations present in the old and absent in the new vintage, deemed c_1, as well as the number of observation present in the new and absent in the old vintage, deemed c_2. Note that we already compared the count of observations of the vintages in Section 4.1.1. However, in this section we pair the observations, which brings additional information. If we denote the count of observations in the old vintage as c_o, in the new vintage — as c_n, and in the join of old and new vintage as c_u, then $c_1 = c_o - c_u$ and $c_2 = c_n - c_u$. If all the observations are paired via the inner join, then $c_o = c_n = c_u$.

Success criteria: The test passes if $c_1 = 0$ and $c_2 = 0$; otherwise the test fails, and the values of c_1 and c_2 are reported. A tester can then assess if the discarded observations appeared in the data are due to normal data churn or because of a defect in data preparation.

4.2. Paired testing

Once we join the old and new vintage (using the approach discussed in Section 4.1.3), we can conduct the following sets of pairwise tests (performing comparisons for a given variable and hierarchy level): 1. the magnitude ratio test, 2. the mean relative error test, 3. the correlation test, and 4. the distribution test. These tests are discussed below.

4.2.1. Magnitude ratios

Rationale: We compare the magnitudes for the minimum, maximum, sum, mean, and median values for each level of hierarchy between the old and new vintages. The expectation is that extreme points of the distribution, as well as the central points, pairwise, should be in the same ballpark, which we will assess by comparing the order of magnitudes.

Method: Let us denote a metric for the ith variable and jth hierarchy level of an old vintage as $m_{i,j,o}$ and for ith variable and jth hierarchy level of a new vintage as $m_{i,j,n}$, respectively. Then the magnitude ratio $R_{i,j}$ is computed as follows:

$$R_{i,j} = \begin{cases} 1, & \text{if } m_{i,j,o} = 0 \text{ and } m_{i,j,n} = 0; \\ \text{undefined}, & \text{if } m_{i,j,o} = 0 \text{ or } m_{i,j,n} = 0; \\ m_{i,j,o}/m_{i,j,n}, & \text{otherwise.} \end{cases} \quad (1)$$

Success criteria: We compute the value of $R_{i,j}$ for each pair of the metrics (min of the old and new vintage, max of the old and new vintage, etc.). If $0.1 < R_{i,j} < 10$ then both values are of the same magnitude and the test succeeds,[7] otherwise —

[6] If machine learning techniques are used subsequently, data scientists may need to rebuild the machine learning model if they want to train the model on all the variables, but this is out of the scope of this paper.

[7] The threshold $R_{i,j}$ can be adjusted on a case-by-case basis, we will further in Section 6.5.

fails and gets reported. Note that we have two special cases. If $m_{i,j,o} = 0$ and $m_{i,j,n} = 0$, then we assume that the magnitudes are identical — setting $R_{i,j} = 1$. If $m_{i,j,o} = 0$ or $m_{i,j,n} = 0$, then we cannot credibly assess magnitude difference; in this case we emit a warning asking an analyst to assess the magnitude difference manually.

Note that given the pairwise nature of the comparison, the ratios of sums and averages will yield identical results. However, we retain both for a practical reason: the sums help an analyst to compare the values of variables at different levels of hierarchies (as, typically, the sum of observations at a lower hierarchy level aggregate to the value at a higher level of the hierarchy) hence the decision to keep the sum values.

4.2.2. Mean relative error

Rationale: The previous test (comparing min, max, etc.) assesses statistics that discard information about pairwise relations of individual observations. Given that we pair observations in the old and new vintage, we can compare each observation using mean relative error. We prefer the mean relative error over the mean absolute error because the values of attributes vary significantly between the variables as well as the variables' hierarchy levels.

Method: Let us pair old and new observations for the ith variable and denote paired vector of observations for the ith variable and jth hierarchy level of old vintage as $x_{i,j,o}$ and for the new vintage as $x_{i,j,n}$. Then the mean relative error $E_{i,j}$ is computed as an average of relative errors of each pair of observations in $x_{i,j,o}$ and $x_{i,j,n}$:

$$E_{i,j} = \langle |(x_{i,j,o} - x_{i,j,n}) \oslash x_{i,j,o}| \rangle, \tag{2}$$

for all non-zero elements of $x_{i,j,o}$, where $\oslash$ is the Hadamard division operator (performing element-wise division of vectors) and $\langle \cdot \rangle$ computes the mean.

By construction, all pairs of observation, where an element from $x_{i,j,o}$ is equal to 0, have to be ignored. If $x_{i,j,o}$ vector has a lot of zero values, then this test may become misleading. In this case one can implement another test of relative change, see the work of Törnqvist et al. (1985) for review and comparison of such tests.

Success criteria: The test considered successful if $E_{i,j} < 0.2$. A smaller value of the threshold can generate a high number of false alarms based on the our previous experiences.

4.2.3. Correlation test

Rationale: We expect that there should be a strong mutual relation between the observations of a given variable in the old and new vintages. To measure the strength of this relation, we compute correlations between the values of a given variable in the old and new vintages. The relation does not necessarily have to be linear but it should be monotonic. Thus, to assess these properties, we use Pearson product-moment correlation coefficient (Pearson, 1896; Myers et al., 2010) (to assess linearity) and Spearman rank-order correlation (Spearman, 1904; Myers et al., 2010) (to assess monotonicity).

Method: We compute Pearson and Spearman correlations coefficients (deemed $r_{i,j}$ and $\rho_{i,j}$, respectively) for pairs of $x_{i,j,o}$ and $x_{i,j,n}$ for each variable i and hierarchy level j. Correlation values range between -1 and 1, with 1 being perfect correlation, -1 — perfect anticorrelation, and 0 — no correlation.

Success criteria: The test is considered successful if $r_{i,j} \geq 0.8$ and $\rho_{i,j} \geq 0.8$, and unsuccessful otherwise (similar to the criteria in Section 4.2.1, this threshold can also be adjusted on a case-by-case basis). From a practical perspective, a lot of real-world variables exhibit nonlinear relations (plus Pearson correlation assumes data normality which is often not the case). Thus, we pay more attention to the case of $\rho_{i,j} < 0.8$ than to the case of $r_{i,j} < 0.8$, because (empirically) they observed that it is a stronger indicator of a defect in the data.

4.2.4. Distribution test

Rationale: The previous test assesses the correlation between the ith variable of the old and the new vintages. In this test, we generalize this approach by comparing distributions of the old and new vintages of this variable. If the distributions are significantly different, then it may be an indicator that there is a defect in the data.

Method: We use the nonparametric two-sample Kolmogorov–Smirnov test (Smirnov, 1939) to compare the differences between the two distributions. The null hypothesis of the test is that the samples are drawn from the same distribution.

Success criteria: The value of the Kolmogorov–Smirnov test p-value for the ith variable and jth hierarchy level is denoted by $S_{i,j}$. If $S_{i,j} < 0.05$, we assume that the null hypothesis is rejected and declare test failure. If $S_{i,j} \geq 0.05$ — the test succeeds (even though it does not imply that the distributions are not different).

4.3. Higher-order testing

The set of higher-order tests (i.e., those that combine the values of the metrics computed in Section 4.2) is composed of the following: 1. the comparison of Spearman correlation coefficients for different levels of hierarchy, 2. hybrid test, and 3. ranking of the number of test failures. The details of the tests are given below.

4.3.1. Comparison of Spearman correlation for different levels of hierarchy

Rationale: In Section 4.2.3, we computed Spearman correlation $\rho_{i,j}$ for ith variable and jth level of hierarchy. Data scientists in our team observed that a significant difference in the ρ values for two adjacent levels of hierarchy (i.e., $\rho_{i,j}$ and $\rho_{i,j+1}$) may indicate a defect in the data of the ith variable. The root cause of such defect often relates to different aggregation procedures (from the raw data) associated with different levels of hierarchy.

Note that while we compute both Pearson and Spearman correlations in Section 4.2.3, the comparison test focuses only on the latter. As we discussed in Section 4.2.3, the $\rho_{i,j} < 0.8$ (Spearman correlation) is a stronger indicator of a defect in the data than $r_{i,j} < 0.8$ (Pearson correlation). Analogously, it was found that comparison of differences in $\rho_{i,j}$ is a better indicator of a defect than a comparison of differences in $r_{i,j}$. Thus, to reduce tester's information overload, it was decided not to include the comparison of $r_{i,j}$ in the report.

Method: We compute relative difference $C_{i,j}$ between two adjacent levels of hierarchy:

$$C_{i,j} = (\rho_{i,j} - \rho_{i,j+1})/\rho_{i,j}, \text{ if } \rho_{i,j} \neq 0. \tag{3}$$

Given J levels of hierarchy, with the 1-st level being the top one and the Jth level being the bottom one, we perform $J - 1$ computations of $C_{i,j}$, with $j = 1, \ldots, J - 1$.

Success criteria: Based on our experience, $-0.1 < C_{i,j} < 0.1$ is considered acceptable. $C_{i,j}$ values outside of this range may indicate a problem with the data of the ith variable and jth or '$j + 1$'th levels of the hierarchy.

4.3.2. Hybrid testing

Rationale: We described multiple tests in the sections above. Intuitively, the higher the number of tests that failed for a given variable and hierarchy level $x_{i,j,n}$ — the higher the chances that there is something wrong with the observations of this variable. We observed that a simultaneous failure of four tests — namely, mean relative error (Section 4.2.2), Spearman and Pearson correlations (Section 4.2.3), and Kolmogorov–Smirnov test (Section 4.2.4) — is a very strong indicator of a defect in the underlying data (based on EA data scientists' past experiences).

Table 2
Example: results obtained from the hybrid test.

Variable name (i)	Hierarchy level (j)	$E_{i,j}$	$r_{i,j}$	$\rho_{i,j}$	$S_{i,j}$
v_5	National	0.578	0.401	−0.278	0.001
v_5	City	0.617	0.672	0.693	0.002
v_8	City	0.669	0.532	0.454	0.046

Table 3
Example: ranking test. To preserve space, a subset of tests is shown in this example.

Variable name (i)	$E_{i,\text{all}}$	$r_{i,\text{all}}$	$\rho_{i,\text{all}}$	$S_{i,\text{all}}$	Total
v_7	4	6	6	6	22
v_3	4	5	2	5	16

Thus, if $x_{i,j,n}$ fails all those tests, it should attract the attention of the data team.

Method: We identify all the variables that failed four above-mentioned tests simultaneously and report them along with the values of the associated metrics. Table 2 displays an example of this report.

Success criteria: As shown in Table 2, a variable's name and corresponding hierarchies are listed in the report if and only if all of the following criteria are satisfied: 1. $E_{i,j} \geq 0.2$, 2. $r_{i,j} < 0.8$, 3. $\rho_{i,j} < 0.8$, and 4. $S_{i,j} < 0.05$.

4.3.3. Ranking of the number of test failures

Rationale: All of the above metrics are computed for each variable and hierarchy level individually. We observed that a test failure at multiple levels of the hierarchy of a given variable acts as a reliable indicator of a defect in the data associated with this variable.

Method: Thus, it is useful to count the number of test failure for each variable and test type and then order them in descending order from the highest number of test failures to the lowest. To reduce clutter, we report only the variables that have at least one test failure associated with them. Example of such ranking is given in Table 3.

Success criteria: An ultimate success is when there are no test failures associated with a variable and this variable does not show up in the report. The higher the number of tests and types of tests that failed — the higher the chances that a variable has a defect in its data.

5. Discussion of tests' properties

5.1. Root causes of test cases' failures

As mentioned at the beginning of Section 4, not every test failure will lead to exposure of a data defect. Instead, a failure suggests that a new vintage is different from the old one in some unexpected way, and that a tester should take a close look at the failure. For the tests operating at a particular hierarchy level (i.e., those discussed in Sections 4.1.2, 4.2, and 4.3.2), a good starting point of an investigation is a review of data transformation procedures for a particular variable and level of hierarchy for which the test case failed. In the case of the test discussed in Section 4.3.1 (examining adjacent levels of hierarchy), the problem typically is associated with data transformation procedures for one of these levels. The root cause of a failure of the test described in Section 4.3.3 often resides in the general procedure that touches multiple levels of the hierarchy of the variable under investigation.

The tests discussed in Section 4.1.1 operate at an even lower level of granularity (as they deal with potentially missing variables or observations). While removal or addition of variables is not uncommon, sometimes an analyst renames a variable by mistake, which often ends up being the root cause for the variable to appear in the report of this test. If the datasets have a significantly different number of observations, it may be caused by datasets truncation or data corruption. A failure of the final metadata-related test, discussed in Section 4.1.3, may indicate corruption of the values in the 'Key' or 'Hierarchy Level' columns.

5.2. Predictive power of tests

As discussed above, not every failure of a test "translates" into an actual defect. However, anecdotally, we observed that higher-order tests described in Sections 4.3.2 and 4.3.3 yield the lowest number of false alerts,[8] followed by the correlation-related tests in Sections 4.2.3 and 4.3.1.

On the other side of the spectrum, the distributions comparison test discussed in Section 4.2.4 yields the highest number of false alarms. This is expected, as the underlying distributions for a large number of variables in periodically updated datasets experience legitimate change to their underlying distributions (which the test detects successfully). However, the list of such variables are typically known to the dataset curators and, thus, can be filtered out with relative ease during the analysis of the report (generated by RESTORE).

The rest of the tests fall in the middle of the spectrum. For example, the change to a distribution also translates into changes to statistics (such as mean, min, and max), which we analyze in Section 4.2.1. However, because we are comparing the magnitudes of these statistics, these tests are less prone to false alarms.

5.3. Data types

All of the tests can process variables to which ratio and, arguably, interval scales (Stevens, 1946) can be applied.

We will also be able to compute the test for numeric variables measured on nominal or ordinal scales (Stevens, 1946), but the results of some of these tests (e.g., magnitude comparison of averages for the ordinal scale) would be questionable from the statistical perspective. Thus, one has to be careful when interpreting the results of the tests.

The test cannot be computed for non-numeric tests, except for the tests discussed in Section 4.1.

Fortunately, curators of datasets typically know data types and measuring scales of the variables in the datasets and can recommend which variables should be excluded from the analysis.

5.4. Assumptions and limitations

Here, we summarize assumptions on which our proposed method is based as well as associated limitations.

1. As discussed in Section 2.4, we assume that all input datasets for RESTORE are in tabular format (or can be transformed to this format). There exist other data formats, e.g., JSON and SQLite. We recommend practitioners to convert non-tabular format datasets into a tabular format before working with RESTORE.
2. We assume that there are no significant changes between two versions of the dataset. However, this might not always be the case. For example, in the case of the current pandemic, geodemographic data can change significantly in certain areas.[9]

[8] We believe that these two tests will also help to mitigate the multiple comparisons problem (Gelman et al., 2012) because we are more interested in the observations that trigger multiple test failures in these two tests.

[9] This may affect the downstream data science pipeline. For example, the subsequent machine learning models may have to be re-trained.

3. If there are attributes that only exist in one vintage dataset, we only report the names of such attributes. As discussed in the section of high-level testing (Section 4.1), our proposed test suite can report missing/new attributes by comparing all the names of attributes in both datasets. However, the tests discussed in Sections 4.2 and 4.3 are not applicable for missing/new attributes in such a circumstance.
4. Our proposed method does not deal with categorical data. As discussed in Section 5.3, we focus on numeric tests. One can leverage our method for categorical data by converting it to numerical data using one-hot encoding (Beck and Woolf, 2000). However, this is beyond the scope of this paper.
5. As discussed in Section 4.2.2, zero values are ignored if they are denominators, because the mean relative errors will mathematically become undefined in such cases. Other formulas — e.g., mean absolute percentage error (MAPE) — may suffer from the same issue. If an analyst experience this issue, they may implement a metric designed to avoid the issue, such as msMAPE (Chen and Yang, 2004) or MASE (Hyndman and Koehler, 2006).
6. The attributes that contain NA values are reported (as discussed in Section 4.1.2). Paired tests in Section 4.2 are not applicable for observations with NA values. Data scientists can manually check the generated report and decide if an observation with NA values require further investigation.

Note that one can create additional metrics for the test suite to address the above-mentioned limitations.

6. The RESTORE package and take-away messages

In this section, we introduce the interface of the RESTORE package in Section 6.1. Then, we validate RESTORE in industrial settings and find answers to our research question in Section 6.2. After that, we discuss threats to validity in Section 6.3. We also discuss potential extensions of RESTORE in Section 6.4. In Section 6.5, we deliver our take-away messages for practitioners applying data validation techniques.

6.1. The interface of RESTORE

We implement the set of tests discussed in Section 4 in an open-source R package, available at Zhang et al. (2021). The installation of the package follows a standard installation process for R package, details are given in the README file of Zhang et al. (2021).

The tests are controlled by a single function `test_two_datasets`. The function ingests old and new vintages of the dataset as well as specification of the hierarchy either from CSV files or from R data frames.

We found that for interactive testing, when a tester adjusted the datasets and wanted to quickly assess the results, the CSV files were more convenient. On the contrary, for automated testing, when the datasets were tested as part of the automated regression test harnesses, the data frame option was more suitable.

The final report is written into a user-specified XLSX file or saved as R data structure (so that it can be easily parsed later, if necessary). Users can select the test which should be stored in the final report. Parameters of the `test_two_datasets` function are as follows.

The parameters `legacy_file` and `target_file` set the path to the files that contain the old vintage of the dataset and the new vintage of the dataset, respectively.

The parameters `hier_pair` sets the path to a CSV file containing 2-tuples 'Parent Hierarchy Level' and 'Child Hierarchy Level', this enables RESTORE to operate on non-linear hierarchies. For example, a tree depicted in Fig. 3(a) will be encoded by 2-tuples shown in Fig. 3(b).

The parameter `hier` points to a CSV file containing an ordered list of hierarchy levels, which is used for sorting the test results in the reports containing hierarchy column (e.g., the one shown in Table 2), see Fig. 3(c) for an example of such a file. Note that this parameter is not used to define the actual hierarchy.

The parameter `thresholds` points to a CSV file containing values for success criteria of tests described in Section 4.

The variables described above have corresponding "twin" parameters (namely, `legacy_df`, `target_df`, `hier_pair_df`, `hier_df`, and `thresholds_df`) which allow to pass the dataset and configuration files in the R data frame format.

The `final_report` parameter specifies the location of the output report file in XLSX format. The parameter `final_data` specifies an output location for the report stored in the R data structure format. The rest of the parameters are used to determine important variable names and a list of tests to run, as summarized in Table 4.

While RESTORE reads data only from CSV files or data frames, it does not imply that we cannot leverage other data formats. We simply need to convert our data into one of these two formats. For example, if the data resides in a relational database, one can issue SELECT SQL query from R using `DBI` (R Special Interest Group on Databases (R-SIG-DB) et al., 2018) package, which will automatically extract and convert the data into the R data frame format.

As part of the package, we provide a sample file demonstrating the usage of RESTORE program interface (see `example.R` on Zenodo Zhang et al., 2021).

6.1.1. Special case: flat hierarchy

To deal with the case of a flat hierarchy (i.e., non-hierarchical dataset), we do not need to pass `hier_pair` and `hier_pair_df` values to `test_two_datasets`. Under the hood, RESTORE adds a dummy hierarchy column to the dataset and runs all the tests against the dataset except for the test comparing correlation coefficients for different values of hierarchy (discussed in Section 4.3.1).

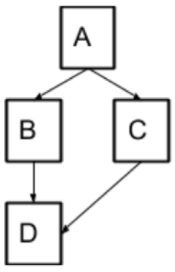
6.2. Validation

The RESTORE R package has been institutionalized into EA's product development cycle. The data scientists use the package to detect defects in the new vintage of the datasets. To assess the benefits of the package, we seek an answer to the following question.

Does the RESTORE package improve the efficiency of the data validation procedure, i.e., does it help identify data errors in less time and with fewer human resources?

First, we quantified the time needed to run all the tests on two reference geodemographic datasets (named D_1 and D_2). The summary statistics for these datasets are shown in Table 5. The table also shows the average and the standard deviation of the execution time of `test_two_datasets` function based on 10 runs of the function for each dataset. We kept the parameter values of the function to defaults, i.e., all of the reports were generated.

Potential data defects have been successfully detected and reported by RESTORE (note that whether a result reported by RESTORE is a true data defect will require manual investigation by data scientists). To preserve space, we will highlight two examples. The examples can also be found in the demo uploaded to Zenodo (Zhang et al., 2021). The first example depicts how Spearman rank-order correlation will list any variable names (as



(a)

Parent Hierarchy Level	Child Hierarchy Level
A	B
A	C
B	D
C	D

(b)

Rank	Hierarchy Level
1	A
2	B
3	D
4	C

(c)

Fig. 3. An example of hierarchy configuration. (a) Example of a non-linear hierarchy. (b) 2-tuple encoding of the non-linear hierarchy from Fig. 3(a). (c) Sample ranking of the hierarchy levels in Fig. 3(a).

Table 4
Additional parameters of `test_two_datasets`.

Parameter	Type	Details	Default value
`key_col`	String	Name of the column containing 'Keys'.	NA
`hier_col`	String	Name of the column containing 'Hierarchy Levels'.	NA
`report_char`	Boolean	TRUE — generate the report of characteristic comparison; FALSE — do not generate the report.	TRUE
`report_na`	Boolean	TRUE — generate the report of missing (NA) variables; FALSE — do not generate the report.	TRUE
`report_discard`	Boolean	TRUE — generate the report of discarded observations; FALSE — do not generate the report.	TRUE
`report_magnitude`	Boolean	TRUE — generate the report of magnitude metrics; FALSE — do not generate the report.	TRUE
`report_mre`	Boolean	TRUE — generate the report of mean relative errors; FALSE — do not generate the report.	TRUE
`report_spearman`	Boolean	TRUE — generate the report of Spearman test; FALSE — do not generate the report.	TRUE
`report_pearson`	Boolean	TRUE — generate the report of Pearson test; FALSE — do not generate the report.	TRUE
`report_distribution`	Boolean	TRUE — generate the report of distribution test; FALSE — do not generate the report.	TRUE
`report_spearman_diff`	Boolean	TRUE — generate the report of Spearman comparison for hierarchies; FALSE — do not generate the report.	TRUE
`report_hybrid`	Boolean	TRUE — generate the report of hybrid metrics; FALSE — do not generate the report.	TRUE
`report_ranking`	Boolean	TRUE — generate the report of ranking; FALSE — do not generate the report.	TRUE

well as all the test results of these two variables, e.g., Pearson correlation) as long as the test results are below the success criteria (set at 0.8). The first example lists five variables (e.g., one of them has a Spearman correlation 0.6607) recorded in the output file. The second example shows a case in the demo where the mean relative error is bigger than the threshold of 0.5, which is 1.2342. There are 20 cases listed in the test of the mean relative error. All of them need to be manually examined by the data scientists.

Our testbed is a laptop equipped with 2 GHz Intel Core i5 CPU and 16 GB memory, running R v.3.5.1 on MacOS v.10.14.3. The datasets are read from files (which is slower than reading the datasets from R data frames). Executing a complete set of tests and generating the final report took, on average, ≈3.4 min for the D_1 and ≈2.3 min for the D_2.

The data scientists also found these tests are computationally inexpensive. Based on the feedback from EA data scientists, the same set of tests, when conducted manually by an experienced data tester takes ≈2 h of the tester's time (per dataset). Thus, using RESTORE speeds up[10] this testing process by ≈97%.

Table 5
Performance evaluation of RESTORE on two reference datasets. Note that the new vintage of D_2 consists of fewer variables compared to the old vintage. Consequently, less computations for comparison are needed — hence less time compared to the one with D_1.

Dataset name	Hierar. levels count	Variables count		Observations count		Time ± St. Dev. (s)
		Old	New	Old	New	
D_1	7	757	762	67,370	67,370	202 ± 11
D_2	7	716	584	67,370	67,370	137 ± 8

[10] Note that we do not take into account the analysis of the test results. However, this time would be identical for both manual-based and RESTORE-based workflows.

The usefulness of RESTORE is also supported by results of an anonymous survey of 15 data scientists in EA who are now using the tool. An anonymous poll was sent to everyone with the following three questions.

1. Is RESTORE helpful? Possible answers were "Extremely useful", "Very useful", "Somewhat useful", "Not so useful", and "Not at all useful".
2. Does RESTORE save time? Possible answers were "Yes" or "No".
3. Does RESTORE identify errors? Possible answers were "Yes" or "No".

One respondent (≈7% of respondents) found RESTORE extremely useful, eleven respondents (≈73% of respondents) — very useful, and three respondents (20% of respondents) — somewhat useful. All fifteen respondents unanimously agreed that RESTORE saves time and identifies errors. The results suggest that the data scientists in EA find RESTORE to be a helpful data validation tool in the data development process because it can efficiently identify data errors.

We have seen the results showing increased efficiency in the identification of potential errors in updated datasets with RESTORE. How does EA benefit from this efficiency in production? RESTORE has been institutionalized by EA and integrated into the dataset development process discussed in Section 2.4. Incremental changes made to the new vintage of the dataset are tested by RESTORE to make sure that the new vintage did not regress. If the tests failed, a root cause detection of the regression is easy to detect, as the failure is typically related to data transformations applied between the two increments.

In the long term, this increased automation and a better approach to data testing will significantly reduce the cost of delivering products to market. We estimate that the time of getting data to market can be reduced by about half (leading to cost reduction and improving customers' satisfaction).

To summarize, the data scientists found these tests to be helpful in practice, i.e., the tests could detect defects in the data reliably. The tests are also computationally inexpensive, which helps to preserve scalability and enable fast verification that the latest changes to a dataset did not inject any new errors.

6.3. Threats to validity

The threats to validity, classified as per (Wohlin et al., 2012; Yin, 2009), are discussed below.

6.3.1. Construct validity

To assess the usefulness of RESTORE, we conducted the survey in EA. We choose an anonymous poll to proactively address the concern that EA data scientists can be positively biased toward the tool that is developed with the help of people from the same organization. We cannot formally prove that the questionnaire is sufficiently detailed to observe and measure bias-inducing factors (e.g., the work culture and politics) that may affect the results of the survey. However, the fact that EA institutionalized RESTORE implicitly supports the results of the survey.

6.3.2. Internal validity

The implementation of RESTORE that we wrote in R may contain bugs. To mitigate this threat, we performed peer code reviews and wrote automated unit tests. EA data scientists performed acceptance testing.

6.3.3. External validity

We cannot claim that our software would be of use for any dataset, which is in line with other software engineering studies, suffering from the variability of the real world (Wieringa and Daneva, 2015). However, RESTORE is helpful for structured hierarchical and non-hierarchical datasets, where at least some of the attributes exist in multiple vintages. These attributes should be either numeric or should be convertible to numeric values (as discussed in Section 5.4).

Moreover, we cannot claim that our list of metrics is exhaustive. To mitigate this threat, we published our source code (as well as a demo program) on GitHub so that anyone can adopt and extend RESTORE for their needs.

6.4. Potential extensions of RESTORE

The current version of RESTORE works by focusing on a pairwise comparison of numerical datasets (measured using ratio and interval scale, as discussed in Section 5.3) that can be loaded into memory. This is sufficient for our use-cases. We released RESTORE as an open-source package so that one can extend or alter the tests implemented in RESTORE based on their specific use-cases or requirements. Below, we sketch potential ways to extend the package if one needs to compare large volumes of data, desires to compare other types of variables, or would like to do non-paired comparison of variables.

RESTORE has been validated in production to process medium size datasets (e.g., datasets that comprise 600+ variables with ≈1.5 million observations). Currently, RESTORE reads all data into memory. This may be an issue for very large datasets (a.k.a. big data). This can be mitigated by altering the process of ingestion datasets into the package: rather than loading the whole dataset into memory, one can process a subset of columns (e.g., loaded using `fread` function from R `data.table` package Dowle and Srinivasan, 2017) in multiple iterations.[11] Alternatively, if the number of observations is such that they cannot be loaded into memory, then one can leverage an external framework, such as Spark, and perform the computations outside of the R engine. Note that Spark integrates into R, e.g., using `sparklyr` package (Luraschi et al., 2018).

If a tester needs to apply RESTORE to other types of data, some of the tests (discussed in Section 5.3) are readily applicable. One can extend the package by adding additional tests. For example, to extend comparison of distributions to ordinal data, one can adopt Mann–Whitney U test (Mann and Whitney, 1947).

Finally, as discussed in Section 4.1.3, we did not perform comparison on non-paired observations (i.e., non-joined ones) of the datasets, as, empirically, they were found less useful for detecting defects in our datasets. However, if one desires to apply the tests to non-paired observations of a given variable, then it can be done with relative ease — all the tests, with the exception of the ones discussed in Sections 4.2.2, 4.2.3, and 4.3.1, are applicable to non-paired data.

6.5. Take-away messages (the best practices)

Based on our hands-on experiences, we introduce five suggestions for data scientists who will develop data regression testing tools. To the best of our knowledge, our proposed best practices are the first general guidelines proposed for data scientists who want to adopt automated data validation in data preparation. In general, a data validation framework for datasets should have five characteristics as follows, and data scientists should adopt best practices to incorporate in the data validation these characters.

[11] Given that computations for every variable are independent of each other, the computations can be easily parallelized using `foreach` (Microsoft and Weston, 2017) and `parallel` (R. Core Team, 2017) packages.

1. **Comprehensive test oracles**. Data scientists should have explicit knowledge of the data structure and the reasonable ranges of values in the datasets. The testing framework will include multiple testing rules to test various aspects that may negatively affect the data quality.
2. **Automated testing**. Compared to manual testing, where a set of tests is performed manually, automated testing significantly reduces the human and time resources for the testing process.
3. **Modularity-driven testing**. Keep in mind the variability of datasets. Thus, there is no one-size-fits-all solution. A scalable testing framework can be altered based on the nature of datasets. The set of tests can be truncated or extended so that all the erroneous data are discovered, and the false alarm rate is kept within a reasonable range (whichever data scientists are comfortable with).
4. **Flexible thresholds**. The thresholds in testing can be set and adjusted based on empirical studies. For example, data scientists can decrease the threshold if the sensitivity is high. Thus, it is essential to keep the thresholds adjustable in the design of the testing framework. In this way, data scientists can always change the testing criterion if needed.
5. **Continuous integration**. In case there are more than one data scientist who is responsible for data testing, or there are more than one data development teams, we would better use the best practices to manage code dependencies. For example, a version control system, such as git (Spinellis, 2012), can be employed to empower people to collaborate. Besides that, we should also aim to ensure the testing framework to run on different environments, e.g., Linux and Mac OS. In such a manner, it is easy to share between team members (or different teams) without complicated configuration.

Our experience can be extended to a more general software testing life cycle (STLC). There are various processes of STLC, they all must include, in some form, the four fundamental activities (Hooda and Chhillar, 2015), i.e., (1) requirement analysis and test analysis, (2) test planning and preparation, (3) test case development and test execution, and (4) test cycle closure. The stages of STLC are activities conducted during the test procedure. Our experience report focuses on some best practices for those who want to adopt automated testing. Those best practices can be performed at any stage of STLC, e.g., automated testing can involve requirement analysis and testing planning.

Note that our data regression testing take-away messages resonate with some best practices of software regression testing in software. However, the former focuses on data defects while the latter investigates software defects. We argue that both, data and software regression testing, are crucial to ensure the overall quality of data-driven software.

7. Related work

Software engineering best practices for data science. Jones (2009) gives an overview of software engineering best practices and introduces 50 best practices based on the study of long lifecycle projects in the industry. Kim et al. (2016) discuss software-oriented data analytics. They conduct a survey on data scientists in Microsoft and propose a set of strategies to help data scientists increase the impact and actionability of their work. Begel and Zimmermann (2014) present the results from two surveys related to data science in the field of software engineering. The results list the most concerned questions from practitioners in 12 categories, including best practices of development and testing practices.

Data validation frameworks. Gao et al. (2016) provided a survey of big data challenges and data validation functionalities in existing big data platforms, such as Microsoft Azure HDInsight (Microsoft, 2021). While there exist various data validation tools, most of them focus on testing a single dataset (e.g., Great Expectations, 2020) rather than comparison of two datasets, which is a key difference between our solution and other data validation tools. Comparing Great Expectations with RESTORE, Great Expectations is not designed for hierarchical data and thus is not suitable for proper testing of geodemographic data. IBM SPSS Statistics (IBM Knowledge Center, 2020) can identify suspicious and invalid cases, variables, and data values in a given dataset. IBM SPSS Statistics focuses on providing a set of data validation rules for a single dataset, e.g., flag incomplete IDs. However, it cannot validate two versions of the same dataset. Thus, this approach is complementary to ours, which automates the checks of two datasets, saving an analyst's time. In the SPSS case, such checks would have to be built from scratch. Bonter and Cooper (2012) developed a data validation protocol for a bird monitoring program. The tool validates and filters the data to remove potential sources of errors and bias. Sadiq et al. (2004) identified the importance of data flow validation in workflow processes. They defined some possible errors in the data flow, e.g., missing data and redundant data. Polyzotis et al. (2017) investigated four common data management challenges (data validation is one of them) in machine learning. Later, this research group in Google (Breck et al., 2019) studied data validation in the machine learning pipeline and identified the importance of detecting data defects early because the model trained with buggy data often amplifies the data bugs over a feedback loop. They proposed a data validation to quantify the distribution distance between training data and new data. However, automated tests are not in the scope of this work. Hence, it is hard to merge their methods or tools into a continuous integration pipeline. The data linter (Hynes et al., 2017) focused on data validation in the ETL pipeline. It is a machine learning-based tool that can automatically inspect input datasets for machine learning models, identify potential data issues (lints), and suggest potentially useful feature transforms for a given model type. The data linter does not compare two versions of datasets for data validation.

There exist some work to ensure data integrity during data migration (Paygude and Devale, 2013a,b; Rathika and Arcokiam, 2014). Instead of comparing two datasets, their approaches detect if the data is altered during the migration process. Thus, we cannot apply their approaches in our case. Below, we also provide some other related but complementary papers.

Database testing frameworks. There exists a significant amount of test frameworks for testing database engines and business logic that alters the data in the databases (Kapfhammer and Soffa, 2003; Maule et al., 2008; Haraty et al., 2002; Nanda et al., 2011; Haftmann et al., 2005). In addition, some database testing frameworks are available (DbFit Community, 2019; DbUnit Community, 2019; NDbUnit Community, 2019; DBTD Project, 2019; Microsoft, 2019). However, none of them are suitable for testing dataset vintages.

Open source projects for regression testing of databases. Regression testing tools for databases try to assure that a query (captured in one of the previous releases) executes successfully (in the release under test). This functionality is available in many existing automated database testing frameworks (DbFit Community, 2019; DbUnit Community, 2019; NDbUnit Community, 2019; DBTD Project, 2019; Microsoft, 2019). However, this will typically be inadequate for our needs as successful execution of a statement cannot guarantee that the returned results are correct (as was discussed in Section 1). Some database testing frameworks, e.g., DbFit Community (2019), can readily check if

the recordsets are identical and highlight the difference between them. However, as we discussed before, changes between vintages of a dataset are expected. Thus, these tests are not sufficient for our needs.

8. Conclusion

In this paper, we focused on applying software engineering best practices to the automation of data validation. Our data under study is taken from the geodemographic domain. We presented a set of tests that enable automated detection of defects in a new vintage of a dataset. We implemented the tests in an open-source R package called RESTORE and validated it in practice. We showed that the adoption of RESTORE can help the procedure of data validation achieve 1. efficiency — reducing the cost to incorporate a new vintage of a dataset, 2. simplicity — encapsulating a batch of relatively complex testing rules into one interface, and 3. scalability — processing datasets comprised of about 1.5 million observations and more than 600 variables. Moreover, we also proposed a set of strategies for the best practices in data validation based on our own experience.

This set of tests is of interest to practitioners, as using the RESTORE package on their datasets gives them the advantages to 1. have more certainty about delivery dates for products, 2. reduce the occurrence of data defects in products, and 3. dedicate more time to developing new functionality, rather than testing the existing one.

CRediT authorship contribution statement

Lei Zhang: Conceptualization, Methodology, Software, Writing – original draft. **Sean Howard:** Project administration, Data curation, Methodology. **Tom Montpool:** Investigation, Methodology. **Jessica Moore:** Project administration. **Krittika Mahajan:** Data curation. **Andriy Miranskyy:** Conceptualization, Software, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Andriy Miranskyy reports financial support, administrative support, and writing assistance were provided by Environics Analytics.

Data availability

The data that has been used is confidential.

Acknowledgments

The work reported in this paper is supported and funded by Natural Sciences and Engineering Research Council of Canada, Ontario Centres of Excellence, and Environics Analytics. We thank Environics Analytics data scientists for their valuable feedback.

References

Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., Zimmermann, T., 2019. Software engineering for machine learning: A case study. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). IEEE, pp. 291–300.

Beck, J., Woolf, B.P., 2000. High-level student modeling with machine learning. In: Intelligent Tutoring Systems, 5th International Conference, ITS 2000, MontrÉAl, Canada, June 19-23, 2000, Proceedings. Springer, Montréal, Canada, pp. 584–593. http://dx.doi.org/10.1007/3-540-45108-0_62.

Begel, A., Zimmermann, T., 2014. Analyze this! 145 questions for data scientists in software engineering. In: 36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014. ACM/IEEE, Hyderabad, India, pp. 12–23. http://dx.doi.org/10.1145/2568225.2568233.

Bonter, D.N., Cooper, C.B., 2012. Data validation in citizen science: a case study from project FeederWatch. Front. Ecol. Environ. 10 (6), 305–307.

Breck, E., Polyzotis, N., Roy, S., Whang, S., Zinkevich, M., 2019. Data validation for machine learning. In: Talwalkar, A., Smith, V., Zaharia, M. (Eds.), Proceedings of Machine Learning and Systems 2019, MLSys 2019, Stanford, CA, USA, March 31 - April 2, 2019. mlsys.org, pp. 334–347, URL: https://proceedings.mlsys.org/book/267.pdf.

Cao, L., 2017. Data science: A comprehensive overview. ACM Comput. Surv. 50 (3), 43:1–43:42. http://dx.doi.org/10.1145/3076253.

Chen, Z., Yang, Y., 2004. Assessing forecast accuracy measures. In: Preprint Series, vol. 2010, pp. 2004–2010.

Columbus, L., 2018. 10 Charts that will change your perspective of big data's growth. https://www.forbes.com/sites/louiscolumbus/2018/05/23/10-charts-that-will-change-your-perspective-of-big-datas-growth/?sh=34a6c0a92926.

DbFit Community, 2019. Dbfit. [Online; accessed 31-January-2019], http://dbfit.github.io/dbfit.

DBTD Project, 2019. DB test driven. [Online; accessed 31-January-2019], http://www.dbtestdriven.com.

DbUnit Community, 2019. Dbunit. [Online; accessed 31-January-2019], http://dbunit.sourceforge.net.

Dowle, M., Srinivasan, A., 2017. data.table: Extension of 'data.frame'. R Foundation for Statistical Computing, URL: https://CRAN.R-project.org/package=data.table, R package version 1.10.4-3.

Dustin, E., Rashka, J., Paul, J., 1999. Automated Software Testing: Introduction, Management, and Performance. Addison-Wesley Professional.

Gao, J.Z., Xie, C., Tao, C., 2016. Big data validation and quality assurance - issuses, challenges, and needs. In: 2016 IEEE Symposium on Service-Oriented System Engineering, SOSE 2016, Oxford, United Kingdom, March 29 - April 2, 2016. IEEE Computer Society, pp. 433–441. http://dx.doi.org/10.1109/SOSE.2016.63.

Gelman, A., Hill, J., Yajima, M., 2012. Why we (usually) don't have to worry about multiple comparisons. J. Res. Educ. Eff. 5 (2), 189–211. http://dx.doi.org/10.1080/19345747.2011.618213, arXiv:https://doi.org/10.1080/19345747.2011.618213.

Goss, J., 1995. "We know who you are and we know where you live": The instrumental rationality of geodemographic systems. Econ. Geogr. 71 (2), 171–198.

Great Expectations, 2020. Great expectations home page. https://greatexpectations.io/.

Haftmann, F., Kossmann, D., Kreutz, A., 2005. Efficient regression tests for database applications. In: CIDR. Citeseer, Asilomar, USA, pp. 95–106.

Haraty, R.A., Mansour, N., Daou, B.A., 2002. Regression testing of database applications. J. Database Manag. 13 (2), 31–42.

Hooda, I., Chhillar, R.S., 2015. Software test process, testing types and techniques. Int. J. Comput. Appl. 111 (13).

Huizinga, D., Kolawa, A., 2007. Automated Defect Prevention: Best Practices in Software Management. John Wiley & Sons, Hoboken, USA.

Hyndman, R.J., Koehler, A.B., 2006. Another look at measures of forecast accuracy. Int. J. Forecast. 22 (4), 679–688.

Hynes, N., Sculley, D., Terry, M., 2017. The data linter: Lightweight, automated sanity checking for ML data sets. In: NIPS MLSys Workshop, Vol. 1.

IBM Knowledge Center, 2020. IBM SPSS statistics — Validate data. https://www.ibm.com/support/knowledgecenter/SSLVMB_23.0.0/spss/data_validation/idh_idd_vdtab_variables.html.

IDC, 2018. DataAge 2025 - the digitization of the world | seagate Canada. https://www.seagate.com/ca/en/our-story/data-age-2025/.

Jones, C., 2009. Software Engineering Best Practices. McGraw-Hill, Inc., New York, NY, USA.

Kapfhammer, G.M., Soffa, M.L., 2003. A family of test adequacy criteria for database-driven applications. In: ACM SIGSOFT Software Engineering Notes, Vol. 28. ACM, Helsinki, Finland, pp. 98–107.

Kelly, D., 2007. A software chasm: Software engineering and scientific computing. IEEE Softw. 24 (6), 118–120. http://dx.doi.org/10.1109/MS.2007.155.

Kim, M., Zimmermann, T., DeLine, R., Begel, A., 2016. The emerging role of data scientists on software development teams. In: Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016. ACM/IEEE, Austin, TX, USA, pp. 96–107. http://dx.doi.org/10.1145/2884781.2884783.

Kim, M., Zimmermann, T., DeLine, R., Begel, A., 2018. Data scientists in software teams: State of the art and challenges. IEEE Trans. Software Eng. 44 (11), 1024–1038. http://dx.doi.org/10.1109/TSE.2017.2754374.

Lewis, W.E., 2000. Software Testing and Continuous Quality Improvement. Auerbach publications, Boca Raton, USA.

Luraschi, J., Kuo, K., Ushey, K., Allaire, J., The Apache Software Foundation, 2018. sparklyr: R Interface to Apache Spark. URL: https://CRAN.R-project.org/package=sparklyr, R package version 0.9.2.

Maier, D., 1983. The Theory of Relational Databases. Computer science Press, Rockville, MD, USA.

Mann, H.B., Whitney, D.R., 1947. On a test of whether one of two random variables is stochastically larger than the other. Ann. Math. Stat. 18 (1), 50–60.

Maule, A., Emmerich, W., Rosenblum, D.S., 2008. Impact analysis of database schema changes. In: Proc. of the 30th Int. Conference on Software Engineering. ACM, Leipzig, Germany, pp. 451–460.

McKinsey Analytics, 2018. Analytics comes of age. https://www.mckinsey.com/~/media/McKinsey/Business%20Functions/McKinsey%20Analytics/Our%20Insights/Analytics%20comes%20of%20age/Analytics-comes-of-age.ashx.

Mell, P.M., Grance, T., 2011. SP 800-145. The NIST Definition of Cloud Computing. Technical Report, National Institute of Standards & Technology, Gaithersburg, MD, USA.

Microsoft, 2019. SQL Server Data Tools. [Online; accessed 31-January-2019], https://visualstudio.microsoft.com/vs/features/ssdt/.

Microsoft, 2021. Cloud computing services | microsoft azure. https://azure.microsoft.com/en-ca/.

Microsoft, Weston, S., 2017. foreach: Provides Foreach Looping Construct for R. R Foundation for Statistical Computing, URL: https://CRAN.R-project.org/package=foreach, R package version 1.4.4.

Myers, J.L., Well, A.D., Lorch, Jr., R.F., 2010. Research Design and Statistical Analysis, third ed. Routledge, New York, NY, US.

Nanda, A., Mani, S., Sinha, S., Harrold, M.J., Orso, A., 2011. Regression testing in the presence of non-code changes. In: Proc. of the 4th IEEE Int. Conf. on Software Testing, Verification and Validation. IEEE, Berlin, Germany, pp. 21–30.

NDbUnit Community, 2019. Ndbunit. [Online; accessed 31-January-2019], https://github.com/NDbUnit/NDbUnit.

Paygude, P., Devale, P., 2013a. Automated data validation testing tool for data migration quality assurance. Int. J. Mod. Eng. Res. (IJMER) 599–603.

Paygude, P., Devale, P., 2013b. Automation of data validation testing for QA in the project of DB migration. Int. J. Comput. Sci. 3 (2), 15–22.

Pearson, K., 1896. Mathematical contributions to the theory of evolution. III. Regression, heredity, and panmixia. Philos. Trans. R. Soc. Lond. Ser. A Contain. Pap. Math. Phys. Charact. 187, 253–318.

Polyzotis, N., Roy, S., Whang, S.E., Zinkevich, M., 2017. Data management challenges in production machine learning. In: Salihoglu, S., Zhou, W., Chirkova, R., Yang, J., Suciu, D. (Eds.), Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017. ACM, pp. 1723–1726. http://dx.doi.org/10.1145/3035918.3054782.

R. Core Team, 2017. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, URL: https://www.R-project.org/.

R Special Interest Group on Databases (R-SIG-DB), Wickham, H., Müller, K., 2018. DBI: R Database Interface. R Foundation for Statistical Computing, URL: https://CRAN.R-project.org/package=DBI, R package version 1.0.0.

Rafi, D.M., Moses, K.R.K., Petersen, K., Mäntylä, M.V., 2012. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In: Proc. of the 7th Int. Workshop on Automation of Software Test. AST '12, IEEE Press, Piscataway, NJ, USA, pp. 36–42.

Rathika, V., Arcokiam, L., 2014. Automated data validation framework for data quality in big data migration projects. SSRG Int. J. Comput. Sci. Eng. 1 (10), 24–27.

Sadiq, S., Orlowska, M., Sadiq, W., Foulger, C., 2004. Data flow and validation in workflow modelling. In: Proceedings of the 15th Australasian Database Conference-Volume 27. Citeseer, pp. 207–214.

Smirnov, N.V., 1939. On the estimation of the discrepancy between empirical curves of distribution for two independent samples. Bull. Math. Univ. Moscou 2 (2), 3–14.

Spearman, C., 1904. The proof and measurement of association between two things. Am. J. Psychol. 15 (1), 72–101.

Spinellis, D., 2012. Git. IEEE Softw. 29 (3), 100–101.

Stevens, S.S., 1946. On the theory of scales of measurement. Science 103 (2684), 677–680.

Törnqvist, L., Vartia, P., Vartia, Y.O., 1985. How should relative changes be measured? Amer. Statist. 39 (1), 43–46.

van Megen, R., Meyerhoff, D.B., 1995. Costs and benefits of early defect detection: experiences from developing client server and host applications. Softw. Qual. J. 4 (4), 247–256.

Wieringa, R.J., Daneva, M., 2015. Six strategies for generalizing software engineering theories. Sci. Comput. Programm. 101, 136–152. http://dx.doi.org/10.1016/j.scico.2014.11.013.

Wirth, N., 2008. A brief history of software engineering. IEEE Ann. Hist. Comput. 30 (3), 32–39. http://dx.doi.org/10.1109/MAHC.2008.33.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A., 2012. Experimentation in Software Engineering. In: Computer Science, Springer Berlin Heidelberg, Berlin, Heidelberg.

Yin, R., 2009. Case Study Research: Design and Methods. In: Applied Social Research Methods, SAGE Publications, Thousand Oaks, CA, USA.

Zhang, L., Howard, S., Montpool, T., Moore, J., Mahajan, K., Miranskyy, A., 2021. RESTORE: Regression testing tool for datasets. http://dx.doi.org/10.5281/zenodo.4601784, https://zenodo.org/record/4601784.

Zhang, Y., Zhang, T., Jia, Y., Sun, J., Xu, F., Xu, W., 2017. DataLab: Introducing software engineering thinking into data science education at scale. In: 39th IEEE/ACM International Conference on Software Engineering: Software Engineering Education and Training Track, ICSE-SEET 2017, Buenos Aires, Argentina, May 20-28, 2017. ACM/IEEE, Buenos Aires, Argentina, pp. 47–56. http://dx.doi.org/10.1109/ICSE-SEET.2017.7.

Lei Zhang is a post-doctoral research fellow of the Computer Science Department at Ryerson University, Toronto, Canada. His ten years of research experience are mostly in software engineering and system modeling. He is particularly interested in understanding the interaction between software engineering and emerging technologies, such quantum computing and machine learning. He has served as a committee member and a reviewer for several international software engineering journals and conferences. Outside of academia, he also has over five years' software development experiences as a software engineer and project coordinator. Zhang received a Ph.D. in computer science from McMaster University, Canada

Sean Howard has 15 years of experience as a data professional. His key areas of expertise in the development of data products for both monetization and internal analytics/insights use cases. He spent the first 12 years of his career at Environics Analytics developing their suite of data products. After leaving Environics Analytics, Sean worked in multiple consulting roles focusing on customer analytics and segmentation. For the last 2 years, Sean has worked at Rogers Sports & Media helping them mature their data, analytics, and insights products for their ad revenue business.

Tom Montpool was the Vice President, Data Production at EA. As one of the original members of EA, and with two decades of experience in the industry ranging from research analyst to product management, and then as Director, Standard Data, Tom had a unique perspective on the production, distribution, and use of EA's data products. Tom had a bachelor's and master's degree in geography from Queen's University and the University of Waterloo, respectively.

Jessica Moore currently leads the Project Management Office at Environics Analytics. She has supported product development and quality assurance activities at EA for nearly 10 years. Jessica has a bachelor's degree in Arts & Business from the University of Waterloo and several certifications from the University of Toronto, Humber College, and Georgian College.

Krittika Mahajan is the Director of Software Quality Assurance at Environics Analytics. She has been building, consulting, and expanding on quality assurance methods at EA for the past five years. She holds a bachelor's degree and master's degree in Chemistry from Panjab University and a Certification in Information Systems Management from the Toronto Metropolitan University.

Andriy Miranskyy is an associate professor in the Department of Computer Science at Ryerson University, Toronto, Canada. His research interests are in mitigating risk in software engineering, with the focus on large-scale software systems. Miranskyy received his Ph.D. in applied mathematics from the University of Western Ontario, London, Canada.