RESTORE: Automated Regression Testing for Datasets

Lei Zhang\textsuperscript{1}, Sean Howard\textsuperscript{2}, Tom Montpool\textsuperscript{3}, Jessica Moore\textsuperscript{4}, Krittika Mahajan\textsuperscript{5}, Andriy Miransky\textsuperscript{6}

\textsuperscript{1,6}Department of Computer Science, Ryerson University, Toronto, Canada
\textsuperscript{2,3}Environics Analytics, Toronto, Canada
\{lei.zhang, avm\}@ryerson.ca, \{sean.howard, tom.montpool, jessica.moore, krittika.mahajan\}@environicsanalytics.com

ABSTRACT
In data mining, the data in various business cases (e.g., sales, marketing, and demography) gets refreshed periodically. During the refresh, the old dataset is replaced by a new one. Confirming the quality of the new dataset can be challenging because changes are inevitable.

How do analysts distinguish reasonable real-world changes vs. errors related to data capture or data transformation? While some of the errors are easy to spot, the others may be more subtle. In order to detect such types of errors, an analyst will typically have to examine the data manually and assess if the data produced are "believable". Due to the scale of data, such examination is tedious and laborious. Thus, to save the analyst's time, it is important to detect these errors automatically. However, both the literature and the industry are still lacking methods to assess the difference between old and new versions of a dataset during the refresh process.

In this paper, we present a comprehensive set of tests for the detection of abnormalities in a refreshed dataset, based on the information obtained from a previous vintage of the dataset. We implement these tests in automated test harness made available as an open-source package, called RESTORE, for R language. The harness accepts flat or hierarchical numeric datasets. We also present a validation case study, where we apply our test harness to hierarchical demographic datasets. The results of the study and feedback from data scientists using the package suggest that RESTORE enables fast and efficient detection of errors in the data as well as decreases the cost of testing.

KEYWORDS
datasets, testing, vintages

1 INTRODUCTION
Datasets are not set in stone; they have to be refreshed periodically in many areas (e.g., sales, census, or marketing). For example, many demographic data and surveys (leveraged in geodemography\textsuperscript{1}) 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 leads to dataset refresh, where the previous version of the dataset is replaced by a newer vintage. Typically, during the refresh, one needs to apply value added methods like imputation, reconciliations, or reprojections to the raw data before loading these data into a business intelligence system. When working with data, there is always a possibility that the raw data have flaws or the data transformations have unintended consequences, leading to loading incorrect data into production systems.

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, the value of the variable would have no physical meaning: a sample report obtained from the new vintage of data may suggest that the average price of a house for area A is $500K. This value is meaningless and can be easily captured by checking the dataset for negative values.

In another 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.

If upon verification, the suspicious results are deemed erroneous, then the data team has to fix the defect, recreate the dataset as necessary, and reload it into the production system. This process repeats itself until all the data-related defects are detected and eliminated. Then the dataset is loaded into a production database, and the product is made available to a customer. This process is laborious and time-consuming. A single iteration (from detecting a defect in the data, to fixing and reloading the dataset), based on the authors’ experience, may take multiple days, significantly delaying the release of the product to a customer.

Thus, our goal is to detect data-related defects automatically, streamline data refresh schedules, and reduce the cost of detecting and fixing the defects.

1.1 Existing solutions
There exists a significant amount of test frameworks for testing database engines and business logic that alters the data in the databases [12, 14, 18, 24, 29]. In addition, some automated database testing frameworks [3–5, 26, 31] are developed to make sure that the previously captured analytics SQL queries execute successfully in the current version.

The problem we aim to solve is complementary, because in our case — once the dataset is loaded into the database — the data stay constant (due to the nature of the analytics workloads, end-users

\textsuperscript{1}Geodemography is an area of market research, specializing in profiling economic and demographic characteristics of geographical areas [11].
do not alter the data in databases). To reach our goal, we need to examine the changes that happen in the data preparation step (before the data are loaded into the database) and flag the erroneous records and variables.

Essentially, we are interested in verifying and validating the data, rather than (1) validating business logic [12, 14, 18, 24, 29] of an application that uses the database as persistent storage or (2) checking if database engine itself performs correctly [2, 16]. That is, we are interested in (1) detecting the changes in the dataset itself and (2) deciding if a given change is expected or not.

1.2 Contributions

In this paper, we present an automated REgreSsion Testing tool for datasets abbreviated as RESTORE for data regression testing. 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 [17, 20]. In our case the functionality is data-centric.

The tool works by comparing the previous vintage of a dataset with a new one and reporting potential issues. The tool leverages statistical analysis and offers a set of comprehensive testing rules for dataset testing. The tool is written in R language [32] (popular in the data science and statistical communities) and is released as an open-source R package on GitHub [35].

We also provide a validation case study based on a sample hierarchical (i.e., tree-like) geodemographic dataset. The study shows that RESTORE can efficiently speed up the testing procedure and reduce the cost of testing. The usefulness of RESTORE is also supported by results of an anonymous survey of 15 data scientists who are now using the tool. RESTORE exhibits the following characteristics.

1. **Efficiency**: Reduces the amount of time and effort required to create a new vintage of dataset, by automatically detecting data discrepancies before the dataset is loaded into a database of a business intelligence system.
2. **Variability**: Supports both flat and hierarchical data structures.
3. **Scalability**: Can process medium size datasets (tested on datasets comprised of 600+ variables with \( \approx 1.5 \) million observations) and can be scaled up to larger datasets.
4. **Simplicity**: Encapsulates a batch of relatively complex testing rules into a simple single-function interface. One can easily use it by providing two datasets (read either from R data frame or from an external file) and a small amount of metadata (see Section 4.1 for details).
5. **Flexibility**: Allows to add new tests and alter the existing ones.

The rest of the paper is structured as follows. Section 2 introduces background information and our proposed method. Section 3 presents the regression tests. Section 4 discusses the interface of RESTORE. Section 5 depicts related work. Finally, Section 6 concludes the paper.

2 REGRESSION TESTING FOR DATASETS

In this section, we first discuss how the geodemographic data are processed and tested in Section 2.1; then we present details of our method in Section 2.2.

### 2.1 Background

Data accuracy, integrity, and quality are becoming more and more crucial to data analytic solutions [7]. However, it is challenging to verify and validate modern datasets because of the large amount and diversity of data. Thus, it is increasingly critical to develop an efficient and effective solution for data testing to assure the accuracy, integrity, and quality of the data.

It is important to understand how the data are developed and tested in a typical data analytics / data science “shop”. 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 Software-as-a-Service (SaaS) approach [25]. These services require datasets to be refreshed multiple times per year. During the refresh, 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 rolled out to some or all levels of geography. 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 a refresh 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’ behaviour (e.g., select a particular geographic area and then run 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 the ones 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 a variety of forms: e.g., errors in raw data, errors in 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.

### 2.2 Our method

Our goal is to detect data defects as early as possible within the development process with a minimal amount of effort so that we can reduce processing time, thereby creating a more optimized and efficient workflow. Namely, we strive to detect anomalies in datasets in the very first step of the process described in Section 2.1.

We propose to detect the erroneous data early (hence reduction of rework) by introducing a novel approach for automated regression testing of the data. To the best of our knowledge, such a data regression testing framework does not exist. This regression testing will help to detect problems automatically (reducing the amount of manual data testing) and early in the development process (reducing the likelihood of data or metadata defects loaded into the staging database). Such early automated detection of the defects would free resources to focus on more complex workloads and scenarios, 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 [34, 41]), 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).

These steps of automated data regression testing are graphically depicted in Figure 1, which shows the following process (discussed in details in Sections 3 and 4).

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 a human- and machine-readable formats.

Let us now look at the details of the tests used to compare the vintages.

### 3 TESTS’ DESCRIPTION

There exist automated database testing frameworks [3–5, 26, 31] to make sure that the analytic statement (captured in one of the previous releases) executes successfully (in the release under test). However, these will typically be inapplicable to our case. As discussed in Section 1.1, the new vintage of a dataset differs from the old one by construction in our case. Thus, the recordsets generated by the analytics database queries will differ from the old vintage to the new vintage.

To address our problem (i.e., regression testing of the modified dataset), a set of tests performs an approximate comparison (rather than exact comparison as done by [3]) of the datasets. Based on the discussions with EA data scientists, we create ten groups of tests, which will be discussed in details below. The data scientists found these tests to be helpful in practice, i.e., the tests were able to reliably detect defects in the data. The tests are also computationally inexpensive, which helps to preserve scalability and enable fast verification of changes to a dataset (in a test-driven-development manner [1], where a regression test suite can be executed quickly to make sure that no new errors were injected with the latest changes).

We also need to define success criteria for these tests. The criteria are defined based on the practical experience of EA data scientists who 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 3.1, 3.2, and 3.3, respectively. Finally, we discuss the power, usage, and limitations of these tests in Section 3.4.

#### 3.1 High-level testing of vintages

An example of a dataset vintage with $N$ variables 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

**Table 1: Example of a dataset vintage; $v_i$ is a variable name.**

| Key | Hierarchy Level | $v_1$ | $\ldots$ | $v_N$ |
|-----|----------------|------|---------|-------|
| 1   | National       | 100  | $\ldots$| 500   |
| 2   | City           | 3    | $\ldots$| 16    |
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ |
| $M$ | National       | 12   | $\ldots$| 124   |

---

2. We provide details of performance measurements in Section 4.2. In a nutshell, RESTORE can compare two vintages of a dataset containing > 500 variables and > 60,000 observations in less than four minutes on commodity hardware.

3. The term variable is synonymous to column or feature, depending on the reader’s background.
of the variables, (2) checking the variables for missing observations, and (3) counting discarded observations.

3.1.1 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.

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 are 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.

3.1.2 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.

3.1.3 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 3.2). We perform the inner join (in the relational algebra sense of the term [22]) 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: EA data scientists 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 observation 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 3.1.1. 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 $c_{on}$, then $c_1 = c_o - c_{on}$ and $c_2 = c_n - c_{on}$. If all the observations are paired via the inner join, then $c_o = c_n = c_{on}$.

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.

3.2 Paired testing

Once we join the old and new vintage (using the approach discussed in Section 3.1.3), 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.

3.2.1 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 $i$-th variable and $j$-th hierarchy level of an old vintage as $m_{i,j,o}$ and for $i$-th variable and $j$-th hierarchy level of a new vintage as $m_{i,j,n}$. Then the magnitude ratio $R_{i,j}$ is computed as follows:

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

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, otherwise — 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.

3.2.2 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 $i$-th variable and denote paired vector of observations for the $i$-th variable and $j$-th 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 $x_{i,j,n}$:

$$E_{i,j} = \frac{1}{|\{x_{i,j,o} \neq x_{i,j,n}\} \cap \{x_{i,j,o} \neq 0\} |} \times \sum_{x_{i,j,o} \neq 0} \frac{|x_{i,j,o} - x_{i,j,n}|}{|x_{i,j,o}|}.$$
for all non-zero elements of $x_{i,j,o}$, where $\odot$ is the Hadamard division operator (performing element-wise division of vectors) and $\langle \rangle$ 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 [40] 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 feedback of practitioners from EA.

3.2.3 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 correlation does not necessarily have to be linear but it should be monotonic. Thus, to assess these properties, we use Pearson product-moment correlation coefficient [28, 30] (to assess linearity) and Spearman rank-order correlation [28, 38] (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. 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, EA data scientists 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.

3.2.4 Distribution test. Rationale: The previous test assesses the correlation between the $i$-th 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 [37] 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 $i$-th variable and $j$-th 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).

3.3 Higher-order testing

The set of higher-order tests (i.e., those that combine the values of the metrics computed in Section 3.2) 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.

### 3.3.1 Comparison of Spearman correlation for different levels of hierarchy. Rationale: In Section 3.2.3, we computed Spearman correlation $\rho_{i,j}$ for $i$-th variable and $j$-th level of hierarchy. EA data scientists observed that a significant difference in the $\rho$ 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 $i$-th 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 3.2.3, the comparison test focuses only on the latter. As we discussed in Section 3.2.3, 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:

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

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

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

### 3.3.2 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. EA data scientists observed that a simultaneous failure of four tests — namely, mean relative error (Section 3.2.2), Spearman and Pearson correlations (Section 3.2.3), and Kolmogorov-Smirnov test (Section 3.2.4) — is a very strong indicator of a defect in the underlying data. 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) $S_{i,j} < 0.05$.

### 3.3.3 Ranking of the number of test failures. Rationale: All of the above metrics are computed for each variable and hierarchy.
level individually. EA data scientists 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.

3.4 Discussion

3.4.1 Root causes of test cases’ failures. As mentioned at the beginning of Section 3, 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 3.1.2, 3.2, and 3.3.2), 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 3.3.1 (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 3.3.3 often resides in the general procedure that touches multiple levels of the hierarchy of the variable under investigation.

The tests discussed in Section 3.1.1 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 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 3.1.3, may indicate corruption of the values in the ‘Key’ or ‘Hierarchy Level’ columns.

3.4.2 Predictive power of tests. As discussed above, not every failure of a test “translates” into an actual defect. However, anecdotal, EA data scientists observed that higher-order tests described in Sections 3.3.2 and 3.3.3 yield the lowest number of false alerts, followed by the correlation-related tests in Sections 3.2.3 and 3.3.1.

On the other side of the spectrum, the distributions comparison test discussed in Section 3.2.4 yields the highest number of false alarms. This is expected, as the underlying distributions for a large number of variables in periodically refreshed 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 test 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 3.2.1. However, because we are comparing the magnitudes of these statistics, these tests are less prone to false alarms.

3.4.3 Data types. All of the tests can process variables to which ratio and, arguably, interval scales [39] can be applied.

We will also be able to compute the test for numeric variables measured on nominal or ordinal scales [39], 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 3.1.

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.

4 INTRODUCTION TO THE RESTORE PACKAGE

In this section, we introduce the interface of the RESTORE package in Section 4.1. Then, we assess RESTORE usefulness based on user’s feedback in Section 4.2. Finally, we discuss potential extensions of RESTORE in Section 4.3.

4.1 The interface of RESTORE

We implement the set of tests discussed in Section 3 in an open-source R package, available at [35]. The installation of the package follows a standard installation process for R package, details are given in the README file of [35].

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 Figure 2a will be encoded by 2-tuples shown in Figure 2b.

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 Figure 2c for an example of such file. Note that this parameter is not used to define the actual hierarchy.

The parameter thresho1ds points to a CSV file containing values for success criteria of tests described in Section 3.

The variables described above have corresponding "twin" parameters (namely, legacy_df, target_df, hier_pair_df, hier_df, and thresho1ds_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 feature 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 [33] 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 in [35]).

4.1.1 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 3.3.1).

4.2 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 two questions:

1. Does RESTORE package make the testing procedure more efficient?
2. Does RESTORE package decrease the resource cost of dataset testing?

The first question is discussed in Section 4.2.1, the second one — in Section 4.2.2.

4.2.1 Does RESTORE package make the testing procedure more efficient? First, we quantified the time needed to run all the tests on two reference geodemographic datasets (named D1 and D2). 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.

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 minutes for the D1 and ≈ 2.3 minutes for the D2.

Based on the feedback from EA data scientists, the same set of tests, when conducted manually by an experience data tester takes ≈ 2 hours of the tester’s time (per dataset). Thus, using RESTORE speeds up4 this testing process by ≈ 97%.

To further understand the benefit of RESTORE, an anonymous poll was sent to 15 EA data scientists 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. Thus, we can conclude that RESTORE is “very useful” (based on the median value of the answers to Q1), as it saves time (Q2) and effectively identifies errors in datasets (Q3). In combination, these results provide an affirmative answer to the first question, suggesting increased efficiency in the identification and remediation of errors in refreshed datasets.

4.2.2 Does RESTORE package decrease the resource cost of dataset testing? RESTORE has been institutionalized by EA and integrated into the dataset development process discussed in Section 2.1. 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

---

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.
4.3 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 3.4.3) 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.

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 package [8]) in multiple iterations\(^7\). 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 [21].

If a tester needs to apply RESTORE to other types of data, some of the tests (discussed in Section 3.4.3) 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 [23].

Finally, as discussed in Section 3.1.3, 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 3.2.2, 3.2.3, and 3.3.1, are applicable to non-paired data.

5 RELATED WORK

There exists a significant amount of test frameworks for testing database engines and business logic that alters the data in the databases [12, 14, 18, 24, 29]. In addition, some database testing frameworks are available [3–5, 26, 31]. However, none of them are suitable for testing dataset vintages. Below, we provide a short summary of related but complementary papers.

Regression testing for database applications with code changes. Haraty et al. [14] and Haftmann et al. [12, 13] focus on black box testing for database applications and leverage existing testing techniques for traditional software on database applications. In our case, the code of the application remains the same, while the underlying data are changing, hence the complementarity.

Regression testing of schema change. Testing for database schema changes is another relevant topic. Maule et al. [24] analyze the impact of database schema changes on database-driven applications. For example, a column can get renamed in a table, breaking existing queries accessing this column. They present an approach for predicting the impact of relational database schema changes upon object-oriented applications. Namely, they propose a technique to extract dependency relationships between applications and database schemas to perform impact analysis. The existing test frameworks (e.g., DbFit [3]) can detect such an error. However, in

---

\(^7\)Given that computations for every variable are independent of each other, the computations can be easily parallelized using `foreach` [27] and `parallel` [32] packages.

---

| Dataset Name | Hierar. Count | Variables Old | Observations Old Count | Time ± St. Dev. (Seconds) |
|--------------|---------------|---------------|-------------------------|--------------------------|
| D1           | 7             | 757           | 67,370                  | 202 ± 11                 |
| D2           | 7             | 716           | 67,370                  | 137 ± 8                  |
our case the schema typically remains constant (rather the data in the tables change); therefore, this work is complementary to ours.

Regression test selection techniques for data-driven applications. In regression testing, test suites can be large, and it can be time-consuming to process all the test cases. Thus, test selection techniques are widely used. Engström et al. [9, 10] report a review of existing regression test selection techniques based on empirical evaluations. Kaphammer and Soffa [18] as well as Willmor and Embury [42] present test criteria, which capture interactions between an application and a database. Nanda et al. [29] introduce a regression test selection technique to selects a subset of existing test cases. This work assumes the presence of non-code changes, such as configuration files of databases. Rogstad et al. [36] present a similarity- and partition-based test case selection approach for database application regression testing. The test cases are generated from classification tree models. Haraty et al. [15] propose a two-phase test selection technique. In phase one, they adopt an impact analysis based on dependencies that exist among the components of database applications. In phase two, they propose two algorithms to reduce the number of test cases. The existing test selection techniques focus on the regression testing for applications rather than the data that these applications ingest. Thus, they are complementary to our work.

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 [3–5, 26, 31]. 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., [3], 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.

6 CONCLUSION

In this paper, we present a set of tests that enable automated detection of defects in a new vintage of a dataset. We implement the tests in an open-source R package called RESTORE. We show that RESTORE can be used to quickly and efficiently detect defects in a new vintage.

We believe that 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.

This work is also of interest to academics, as it can serve as a building block in the movement of bringing lightweight software engineering practices into the data science realm to improve the quality of data-science-related products [6, 19].

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 Enviroincs Analytics. We thank Enviroincs Analytics data scientists for their valuable feedback.

REFERENCES

[1] Kent Beck. 2003. Test-driven development: by example. Addison-Wesley Professional, Boston, US.
[2] Enzo Cialini, Anthony Loreto, and David Godwin. 2007. Method, system, and program for determining discrepancies between database management systems. US Patent App. 11/262,840.
[3] DbFit Community. 2019. DbFit. http://dbfit.github.io/dbfit. [Online; accessed 31-January-2019].
[4] DbUnit Community. 2019. DbUnit. http://dbunit.sourceforge.net. [Online; accessed 31-January-2019].
[5] Nd4JUnit Community. 2019. Nd4JUnit. https://github.com/Nd4JUnit/Nd4JUnit. [Online; accessed 31-January-2019].
[6] Lito Perez Cruz. 2017. When Data Science Becomes Software Engineering. In Proc. of the 9th Int. Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (KEOD 2017). INSTICC, Funchal, Portugal, 226–232.
[7] Dursun Delen and Haluk Demirkan. 2013. Data, information and analytics as services. Decision Support Systems 55, 1 (2013), 359–363.
[8] Matt Dowle and Arun Shivnivasan. 2017. data.table: Extension of ‘data.frame’. R Foundation for Statistical Computing. https://CRAN.R-project.org/package=data.table R package version 1.10-4-3.
[9] Emelie Engström, Pet Runeson, and Mats Skoglund. 2010. A systematic review on regression test selection techniques. Information and Software Technology 52, 1 (2010), 14–30.
[10] Emelie Engström, Mats Skoglund, and Per Runeson. 2008. Empirical evaluations of regression test selection techniques: a systematic review. In Proc. of the 2nd ACM/IEEE int. symp. on empirical software eng. and measurement. ACM, 22–31.
[11] Jon Goss. 1995. “We Know Who You Are and We Know Where You Live”: The Instrumental Rationality of Geodemographic Systems. Economic Geography 71, 2 (1995), 171–198.
[12] Florian Haftmann, Donald Kossmann, and Alexander Kreutz. 2005. Efficient Regression Tests for Database Applications. In CIDR. Citeseer, Asokean, US, 95–106.
[13] Florian Haftmann, Donald Kossmann, and Eric Lo. 2007. A framework for efficient regression tests on database applications. The VLDB Journal 16, 1 (2007), 145–164.
[14] Ramzi A Haraty, Nashat Mansour, and Bassel A Doua. 2002. Regression testing of database applications. Journal of Database Management 13, 2 (2002), 31–42.
[15] Ramzi A Haraty, Nashat Mansour, and Bassel A Doua. 2004. Regression test selection for database applications. In Advanced Topics in Database Research, Volume 3. IGI Global, US, 141–165.
[16] Thomas E Huxit and Kerry C Tenberg. 1996. Method of checking index integrity in a DB2 database. US Patent 5,579,515.
[17] Dorota Huizinga and Adam Kolawa. 2007. Automated defect prevention: best practices in software management. John Wiley & Sons, Hoboken, US.
[18] Gregory M Kaphammer and Mary Lou Soffa. 2003. A family of test adequacy criteria for database-driven applications. In ACM SIGSOFT Software Engineering Notes, Vol. 28. ACM, Helsinki, Finland, 98–107. Issue 5.
[19] Deanne Larson and Victor Chang. 2016. A review and future direction of agile, business intelligence, analytics and data science. International Journal of Information Management 36, 5 (2016), 700–710.
[20] William E Lewis. 2000. Software testing and continuous quality improvement. Auerbach publications, Boca Raton, US.
[21] Javier Luraschi, Kevin Kuo, Kevin Ushey, JJ Allaire, and The Apache Software Foundation. 2018. sparklyr: R Interface to Apache Spark. R Foundation for Statistical Computing. https://CRAN.R-project.org/package=sparklyr R package version 0.9.2.
[22] David Maier. 1983. The theory of relational databases. Computer science press, Rockville, MD, USA.
[23] Henry B Mann and Donald R Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. The annals of mathematical statistics 18, 1 (1947), 50–60.
[24] Andy Maule, Wolfgang Emmerich, and David S Rosenblum. 2008. Impact analysis of database schema changes. In Proc. of the 9th Int. Conference on Software Engineering. ACM, Leipzig, Germany, 451–460.
[25] Peter M. Mell and Timothy Grance. 2011. SP 800-145. The NIST Definition of Cloud Computing. Technical Report. National Institute of Standards & Technology, Gaithersburg, MD, United States.
[26] Microsoft. 2019. SQL Server Data Tools. https://visualstudio.microsoft.com/vs/features/ssdt/. [Online; accessed 31-January-2019].
