Data Smells in Public Datasets

Shome, Arumoy; Cruz, Luis; Deursen, Arie Van

DOI
10.1145/3522664.3528621

Publication date
2022

Document Version
Final published version

Published in
Proceedings - 1st International Conference on AI Engineering - Software Engineering for AI, CAIN 2022

Citation (APA)
Shome, A., Cruz, L., & Deursen, A. V. (2022). Data Smells in Public Datasets. In *Proceedings - 1st International Conference on AI Engineering - Software Engineering for AI, CAIN 2022* (pp. 205-216). (Proceedings - 1st International Conference on AI Engineering - Software Engineering for AI, CAIN 2022). IEEE. https://doi.org/10.1145/3522664.3528621

Data Smells in Public Datasets

Arumoy Shome
Delft University of Technology
Netherlands
a.shome@tudelft.nl

Luís Cruz
Delft University of Technology
Netherlands
l.cruz@tudelft.nl

Arie van Deursen
Delft University of Technology
Netherlands
arie.vandeursen@tudelft.nl

ABSTRACT

The adoption of Artificial Intelligence (AI) in high-stakes domains such as healthcare, wildlife preservation, autonomous driving and criminal justice system calls for a data-centric approach to AI. Data scientists spend the majority of their time studying and wrangling the data, yet tools to aid them with data analysis are lacking. This study identifies the recurrent data quality issues in public datasets. Analogous to code smells, we introduce a novel catalogue of data smells that can be used to indicate early signs of problems or technical debt in machine learning systems. To understand the prevalence of data quality issues in datasets, we analyse 25 public datasets and identify 14 data smells.

ACM Reference Format:
Arumoy Shome, Luís Cruz, and Arie van Deursen. 2021. Data Smells in Public Datasets. In *1st Conference on AI Engineering - Software Engineering for AI (CAIN'22), May 16–24, 2021, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3522664.3528621

1 INTRODUCTION

Data analysis is a critical and dominant stage of the machine learning lifecycle. Once the data is collected, most of the work goes into studying and wrangling the data to make it fit for training. A highly experimental phase follows where a model is selected and tuned for optimal performance. The final model is then productionised and monitored constantly to detect data drifts and drop in performance [8, 23, 43, 46].

When compared to traditional software, the feedback loop of a machine learning system is longer. While traditional software primarily experiences change in *code*, a machine learning system matures through changes in *data, model & code* [43]. Given the highly tangled nature of machine learning systems, a change in any of the stages of the lifecycle triggers a ripple effect throughout the entire pipeline [46]. Testing such changes also becomes challenging since all three components need to be tested. Besides the traditional test suites, a full training-testing cycle is required which incurs time, resource and financial costs. The surrounding infrastructure of a machine learning pipeline becomes increasingly complex as we move towards a productionised model. Thus catching potential problems in the early, upstream phase of data analysis becomes extremely valuable as fixes are faster, easier and cheaper to implement.

AI has had a significant impact on the technology sector due to the presence of large quantities of unbiased data [38]. But AI's true potential lies in its application in critical sectors such as healthcare, wildlife preservation, autonomous driving, and criminal justice system [12]. Such high-risk domains almost never have an existing dataset and require practitioners to collect data. Once the data is collected, it is often small and highly biased. While AI research is primarily dominated by model advancements, this new breed of *high-stakes AI* supports the need for a more data-centric approach to AI [29, 42, 54].

Since the study of software systems with machine learning components is a fairly young discipline, resources are lacking to aid practitioners in their day-to-day activities. The highly data-driven nature of machine learning makes data equivalent to code in traditional software. The notion of code smells is critical in software engineering to identify early indications of potential bugs, sources of technical debt and weak design choices. Code smells have existed for over 30 years. A large body of scientific work has catalogued the different smells, the context in which they occur and their potential side-effects. To the best of our knowledge, such a catalogue however does not exist for data science.

The research questions along with the contributions of this paper are listed below.

- **RQ1. What are the recurrent data quality issues that appear in public datasets?**
  Analogous to code smells, we introduce the notion of data smells. Data smells are anti-patterns in datasets that indicate early signs of problems or technical debt.
- **RQ2. What is the prevalence of such data quality issues in public datasets?**
  We create a catalogue of 14 data smells by analysing 25 popular public datasets[1]. The catalogue also presents real-world examples of the smells along with refactoring suggestions to circumvent the problem.
  Additionally, we plan to publish the catalogue online under the creative commons license in hopes that students and practitioners find it valuable.

The remainder of the paper is structured as follows. Section 2 provides an overview of related concepts and prior work that has been done. The methodology followed by this paper is presented in Section 3 followed by the results in Section 4. The paper concludes with a discussion of the results, limitations and future work in Section 5, 6 and 7 respectively.

2 RELATED WORK

This section provides an overview of relevant prior work in *code smells*, *data validation* and *AI engineering*.

CAIN'22, May 16–24, 2021, Pittsburgh, PA, USA
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9275-4/21/05.
https://doi.org/10.1145/3522664.3528621

[1] Our analysis of the datasets can be found on Figshare https://figshare.com/s/fd608796dd65f0808e7e

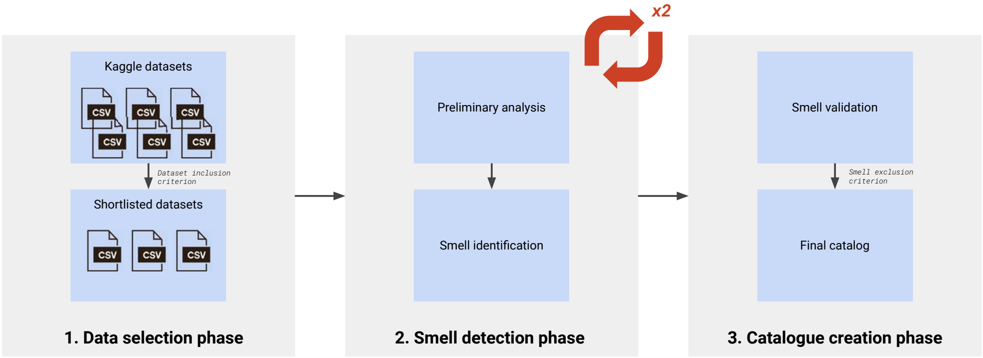


Figure 1: Overview of scientific process

Code smells were originally proposed by Kent Beck in the 1900s and later popularised by *Fowler* in his book **Refactoring** [16, 17]. Code smells are indications of potential problems in the code and require engineers to investigate further. Common code smells include presence of bloated code such as *large classes* & *long methods*, redundant code such as *duplicate code* & *dead code paths* and excessive coupling such as *feature envy* [17, 19]. Code smells have been widely adopted by the software engineering community to improve the design and quality of their codebase. The notion of code smells has also been extended to other areas such as testing [6, 48, 49], bug tracking [50], code review [15] and database management systems [13, 37, 47]. Code smells however still suffer from the problem of lacking generalisability over a large population as most smells are subjective to the developer, team or organisation.

Data validation is a well established field of research with roots in Database Management Systems (DBMS). With the wide adoption of data-driven decision-making by businesses, significant efforts have been made towards automated data cleaning and quality assurance [11, 22, 28, 44]. In the context of machine learning, several tools and techniques have been proposed for improving data quality and automated data validation [7, 9, 27, 32, 40, 55]. *Hynes et al.* present a data linting tool in the context of *Deep Neural Networks (DNNs)*. The tool checks the training data for potential errors both at the dataset and feature level. The paper presents empirical evidence of applying the linter to over 600 open source datasets from Kaggle, along with several proprietary Google datasets. The results indicate that such a tool is useful for new machine learning practitioners and educational purposes [24]. Although there is some overlap between the data linter by Hynes et al. and our data smells project, We argue that Hynes et al. did not follow a systematic approach to collect the linting rules. Our work is complimentary to data linters as our approach exhaustively extracts potential data quality issues from datasets. Our catalogue of data smells can be seen as a framework for systematically extending, or creating new data linting and validation tools.

AI engineering is a relatively young discipline of software engineering (SE) research. The primary focus of the field is to compare and contrast machine learning systems to traditional software systems and adopt best practices from the SE community. The seminal paper by *Sculley et al.* was the first to recognise that machine learning systems accumulate technical debt faster than traditional software [46]. This accelerated rate of technical debt accumulation is due to the highly tangled nature of machine learning models to its data. Machine learning is data-centric as each problem—which requires new or combination of existing datasets—needs to be addressed individually [4, 5, 8, 26, 46, 52].

Data scientists spend the majority of their time working with data, yet unlike in software engineering, lack tools that can aid them in their analysis [8, 23]. This study proposes a catalogue of data smells that can be beneficial to practitioners and used as a framework for development of tools in the future.

3 METHODOLOGY

Figure 1 presents an overview of the scientific process followed in this study. The methodology can be divided into three distinct phases which are presented in more detail below.

3.1 Data selection phase

This study uses Kaggle—an online data repository—to discover datasets for the analysis [2]. All public datasets available on Kaggle are sorted by the *Most Votes* criteria and 25 datasets are shortlisted based on the inclusion criterion presented in Table 2. The sample of datasets only includes CSV files of size smaller than 1GB to facilitate the analysis on a personal laptop. Analysis of unstructured datasets such as text corpus, images, videos and audio is excluded as this calls for specialised tools, additional time and effort which was deemed beyond the scope of this study. Structured datasets on the other hand are more commonly occurring. Practitioners and academics frequently work with structured datasets that are often used for educational purposes. Therefore, we invested our efforts in analysing structured datasets to make our work relevant to a larger demographic. The 25 datasets selected for this study are listed in Table 1. The table also includes additional metadata such as the *size, number of rows and columns, number of votes* and the *latest version* at the time of analysis.

3.2 Smell detection phase

[2] https://www.kaggle.com

Table 1: Selected Datasets

Name	Description	Size	Rows	Columns	Votes	Version
abalone	Predicting age of abalone from physical measurements	188K	4177	9	99	3
adult	Predicting whether income exceeds $50K/yr based on census data	3.8M	32561	15	475	3
airbnb	Airbnb listings and metrics in NYC, NY, USA	6.8M	48895	16	2502	3
avocado	Historical data on avocado prices and sales volumn in Multiple US markets	1.9M	18249	13	2770	1
bitcoin	Bitcoin data at 1-min intervals from select exchanges, Jan 2012 to March 2021	303M	4857377	8	2876	7
breast-cancer	Predict whether the cancer is benign or malignant	123K	569	33	2537	2
comic-dc	FiveThirtyEight DC comic characters	1.1M	6896	13	2465	111
comic-marvel	FiveThirtyEight Marvel comic characters	2.3M	16376	13	2465	111
covid-vaccine	Daily and total vaccination for COVID-19	11M	53595	15	1978	234
covid-vaccine-manufacturer	Vaccinations for COVID-19 by manufacturer	793K	19168	4	1978	234
earthquake	Date, time and location of all earthquakes with magnitude of 5.5 or higher	2.3M	23412	21	435	1
fraud	Anonymised credit card transactions labeled as fraudulent or genuine	144M	284807	31	8775	3
happiness	Happiness scored according to economic production, social support, etc	8.7K	156	9	3401	2
heart	UCI heart disease dataset	12K	302	14	5601	1
insurance	Insurance forecast by using linear regression	55K	1338	7	1621	1
iris	Classify iris plants into three species	4.5K	150	5	2779	2
netflix	Listings of movies and tv shows on Netflix	3.3M	8807	12	6155	5
permit	San Francisco building permits	76M	198900	37	194	1
playstore	Google play store apps data	1.3M	10841	13	47	1
student	Marks secured by students in various subjects	71K	1000	8	3050	1
suicide	Suicide rates overview 1985 to 2016	2.6M	27820	12	2766	1
telco	Telco customer churn	955K	7043	21	1902	1
vgsales	Video game sales	1.3M	16598	11	4248	2
wine	Red wine quality	11K	178	14	1918	2
youtube	Trending YouTube video statistics	60M	40949	16	4381	115

Table 2: Inclusion criterion for datasets

Key	Inclusion Criteria
IC1	Dataset is of *CSV* format.
IC2	Dataset is smaller than 1GB in size.
IC3	Dataset contains structured data.
IC4	Dataset primarily contains numerical and categorical features.

We used the Python programming language [3] along with the data analysis package Pandas [4] to perform the analysis. The smells are identified using a two pass technique which is manually conducted by the first author. The first pass focuses on identifying characteristics of datasets that are indicative of a smell. Since the catalogue of smells evolved as more datasets were analysed during the first pass, a second pass is used to validate the original smells observed in the datasets. The second pass also helps to identify newer smells which were missed in the older datasets during the first pass.

The first pass conducts a preliminary analysis of the datasets listed below. This is a standard list of checks performed by data scientists during data understanding under the CRISP-DM model of data mining [25, 34, 41, 45].

(1) Reading the accompanying data documentation when available.

[3] https://www.python.org
[4] https://pandas.pydata.org

Table 3: Exclusion criterion for smells

Key	Exclusion Criteria
EC1	Smell is not generalisable to structured datasets.
EC2	Smell is not generalisable to other programming languages and tools.

(2) Analysing their *head* and *tail*—both in its entirety and on a feature-by-feature basis.
(3) Observing the column headers and datatypes for relevant meta data such as the expected schema of the dataset.
(4) Analysing the descriptive statistics of the dataset.
(5) Checking for missing values and duplicate rows.
(6) And finally checking correlations amongst features.

We did not analyse the distribution of the features and their relationship with one another (besides checking for correlation). This is because the insights gained from distributional and relational analysis of a particular dataset are not generalisable to other datasets and domains. This is touched upon in more detail in Section 6.

3.3 Catalogue creation phase

We further prune the list of smells using the exclusion criterion listed in Table 3 and additional validation from the second author. Smells which cannot be generalised to other structured datasets are removed. Similarly, smells which are relevant only when using a specific programming language and tools (such as Python and Pandas) but not applicable when using a different toolset (such as Matlab [5], R [6] or Julia [7]) are excluded.

4 RESULTS

This section presents the results obtained from the analysis of public datasets. The most recurrent data quality issues are presented first. A catalogue of data smells showing the prevalence of such data quality issues is presented next (See RQ1 and RQ2 in Section 1). This study analysed **25** public datasets from which **14** data smells were discovered. We group the smells into 4 distinct categories based on their similarity as listed below.

(1) **Redundant value smells** or smells which occur due to presence of features that do not contribute any new information.
(2) **Categorical value smells** or smells which occur due to presence of features containing categorical data.
(3) **Missing value smells** or smells which occur due to absence of values in a dataset.
(4) **String value smells** or smells which occur due to presence of features containing string type data.

Additionally, three more smells were found which could not be grouped into the above categories and are put under the **Miscellaneous smells** category.

Table 4 presents an overview of all data smells along with their distribution. The remainder of this report frequently refers to these smells by their unique key which is also provided here. The *redundant and categorical value smells* are the most common categories with a total occurrence of 33 and 17 respectively. The *missing & string value smells* are the least common categories with a total occurrence of 13 and 12 respectively. *red-corr* is the most common smell, observed in 19 of the 25 datasets. The remaining top five smells include *cat-hierarchy, miss-null, red-uid and misc-unit* which are observed in more than 10 datasets. The least frequently observed smells include *misc-balance, str-human, misc-sensitive, miss-sp-val and miss-bin* which are observed in less than 5 datasets.

Table 4: List of smells

Key	Name	Count
Redundant value smells (red)		33
red-corr	Correlated features	19
red-uid	Unique identifiers	11
red-dup	Duplicate examples	3
Categorical value smells (cat)		17
cat-hierarchy	Hierarchy from label encoding	12
cat-bin	Binning categorical features	5
Miscellaneous value smells (misc)		14
misc-unit	Unknown unit of measure	9
misc-balance	Imbalanced examples	3
misc-sensitive	Presence of sensitive features	2
Missing value smells (miss)		13
miss-null	Missing values	11
miss-sp-val	Special missing values	1
miss-bin	Binary missing values	1
String value smells (str)		12
str-num	Numerical feature as string	5
str-sanitise	Strings with special characters	5
str-human	Strings in human-friendly formats	2

Finally we also analyse the distribution of smells within the datasets. Figure 2 shows a two dimensional histogram of the smells and datasets such that the intersection of datasets where a particular smell occurred is filled. The histogram is colour-coded based on the smell category which allows us to observe the most common smell categories at a glance. The figure also contains two marginal plots across the x and y axes. The marginal plot along the x axis presents a count of smells ***within*** each dataset such that we can identify the datasets with the most and least number of smells. Similarly, the marginal plot along the y axis presents a count of smells ***across*** all datasets such that we can identify the most and least frequently occurring smells.

The remainder of this section presents the smell groups and their corresponding smells in more detail. We present examples of the smells discovered along with an explanation of the underlying problems that may arise. Where applicable, potential strategies to mitigate the problem, context in which the smells may not apply and references from literature (both scientific and grey) are also presented.

[5] https://www.mathworks.com/products/matlab.html
[6] https://www.r-project.org/
[7] https://julialang.org/

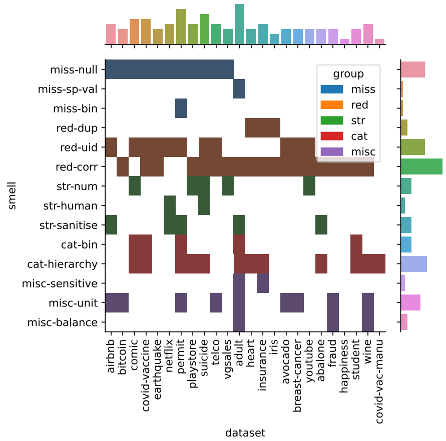


Figure 2: Joint distribution of smells and datasets

4.1 Redundant Value Smells

This section presents smells that indicate presence of redundant information in datasets. This was the most prominent group of smells as 33 occurrences were observed in this study.

4.1.1 Correlated features (red-corr). This study identified 19 datasets that contained correlated features.

Checking for correlation amongst features is a common practise in data science. The correlation between two numerical features is a representation of the linear relationship between them. From a machine learning perspective, presence of uncorrelated features indicates that the features impart new information. But the presence of correlated features is a smell for redundant information.

Datasets come in all shapes and sizes, often containing rows or columns which do not offer new or valuable information. The model training stage usually helps identify features which do not need to be included in the training data, allowing us to engineer a more efficient dataset. Engineering efficient datasets is important since the machine learning lifecycle consists of several deeply coupled stages. Naturally, any form of optimisation—no matter how small—propagates through the downstream stages. A small dataset is easier to understand, faster to train a model on and takes up less storage. The benefits of a small dataset are appreciated especially during the model development phase where many experiments along with their accompanying dataset, model and code are versioned and stored [5, 43].

Presence of correlated features gives practitioners the opportunity to perform feature selection and drop redundant features which do not affect the model's performance.

4.1.2 Unique identifiers (red-uid). This study identified 11 datasets that contained columns containing a unique identifier (uid) for each example in the dataset. For example, the *youtube, earthquake, netflix, telco and avocado* datasets all contain a column carrying a uid for the examples.

Relational databases are the most popular type of databases being used today. Such databases have a column containing a unique identifier (uid) commonly referred to as the *primary key*. Machine learning pipelines are often automated and perform end-to-end operations, starting with data consumption from large data warehouses and data lakes, all the way to publishing a trained machine learning model in production. Although uids are useful when performing merge or join operations on two or more database tables, they become redundant when training machine learning models. Their presence in a dataset is a smell for potential problems in downstream stages.

A machine learning model may learn some hidden relationship between the uids and the target values that produces a high accuracy during training. Such an insight however prevents the model from learning relationships and trends that are generalisable to unseen data and limits its ability to provide meaningful predictions. Furthermore, uids may also prevent the detection of duplicate examples in a dataset. This is another smell that was discovered in this study and discussed further in Section 4.1.3 below.

Although features containing uids in general should not be included in the training set, they can sometimes provide valuable insights during the data analysis stage. The *airbnb* dataset contains the `host_id` and `id` features containing uids for the hosts and the properties respectively. One may regard them as redundant feature and drop them. However further analysis of the columns may lead to interesting insights. For instance, the `host_id` column contains duplicate entries which presents the insight that hosts may own multiple properties. This can further be engineered into a new feature which may help during training. Analysing the columns together can help detect truly duplicate examples (rows with the same property and host id) and outliers (rows with the same property id but different host ids).

4.1.3 Duplicate examples (red-dup). This study identified three datasets, namely *heart, insurance and iris*, that contained duplicate rows. We ignore timeseries data where an event can occur several times resulting in duplicate rows.

Duplicate examples in a dataset are defined as two or more rows which refer to the same entity. They do not serve any purpose and can be removed from the dataset, making their presence a smell for redundancy in the dataset.

Duplicate examples make a dataset bloated. They do not contribute any new information during the data analysis stage. Furthermore, training a machine learning model with a dataset containing duplicate examples can impede the model's performance on unseen data. Training a model using duplicate examples might lead to *overfitting* as it may learn once from the original example and then again from the duplicate example(s).

4.2 Categorical Value Smells

This section presents smells that arise from the presence of categorical data. This study found 17 occurrences of smells from this group.

4.2.1 Hierarchy from label encoding (cat-hierarchy). The values of the `education` feature in the *adult* dataset have a clear hierarchy amongst themselves. Figure 3 shows the probability density plot of the education levels of adults conditioned on their income class. For the given dataset, the probability that an adult earns more given

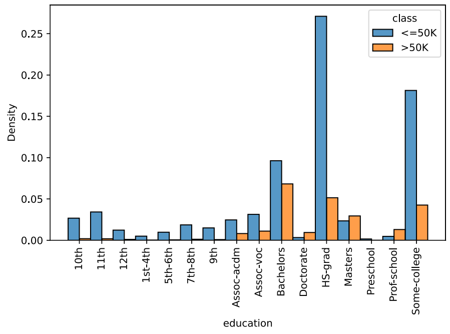


Figure 3: Probability density plot of adult education based on income

that they have better education is higher. We can expose the hierarchy amongst the education levels by assigning a number starting from 0 in ascending order such that higher levels of education are assigned a higher numerical value. This encoding scheme can aid a machine learning model to accurately predict the income class of an individual. Using the same encoding scheme for the `sex` and `race` features in the same dataset can however lead to biased outcomes.

An important characteristic of categorical data is the notion of hierarchy amongst its values. *Label or dummy encoding* is a common technique used in data science to encode categorical data as numbers. This technique preserves the hierarchy amongst the values which may impart useful information to the model during training. Some categorical features however contain sensitive information (sensitive features are discussed in more detail in Section 4.3.1) and do not have hierarchy amongst their values. Label encoding such features can introduce bias into a machine learning model and affect its performance. The presence of sensitive categorical features are therefore a smell to avoid introducing bias into the model.

Label encoding sensitive categorical features can introduce unwanted hierarchy amongst the values and lead to incorrect and biased results in machine learning models. The model may incorrectly associate a sex or race with a higher numerical value to be superior to other values with a lower number. This can be avoided using the *one-hot encoding* technique as opposed to label encoding [2].

4.2.2 Binning categorical features (cat-bin). Figure 4 presents the distribution of the `neighbourhood` feature in the *airbnb* dataset. It is a categorical feature that contains over 200 unique values but several values are rare and do not occur that often. Another example is found in the *adult* dataset where the `native-country` feature contains 42 unique values.

One-hot encoding a feature with high cardinality can result in a very large feature space and incur higher memory, disk space and computation costs throughout the machine learning lifecycle. Presence of categorical features with high cardinality in their data is a smell to perform potential data transformations to reduce the cardinality.

A common practise amongst data scientists to address such a problem is to bin several values together. For example, the `native-county` values can be binned into the seven continents. As an alternative to the `neighbourhood` feature, the *airbnb* dataset also contains the `neighbourhood_group` feature which bins the neighbourhood values into 5 broader areas.

4.3 Miscellaneous Smells

This section presents three smells which did not fit into groups presented earlier. This study identified 14 occurances of smells from this group.

4.3.1 Presence of sensitive features (misc-sensitive). The *adult* dataset presents census information of individuals from 1994. Amongst others, the dataset contains information regarding the *sex, race & income* of individuals. Figure 5a presents the probability density plot of the income class conditioned on the race and sex of individuals. We see that for this dataset, the probability that a male of lighter skin earns more than their female and darker skin counterpart, is significantly higher.

Not all features contribute equally towards knowledge, whether it be during the analysis or model training. Section 4.1 motivated the need to remove redundant features from a dataset, leaving behind *high-impact* features that contribute the most towards analysis and model training. While most high-impact features lead to interesting insights during analysis, not all should be used to train machine learning models. Presence of high-impact features are a smell to identify sensitive features that may lead to biased and unfair model predictions.

Going back to the example presented above, a machine learning model trained and tested on this dataset would be able to predict the income class of an individual with high accuracy. However such a model when used in production to making business decisions will lead to unfair and biased predictions given it was trained with historical data with similar traits [1, 36, 53]. Use of biased models for predictive policing and criminal justice systems can have far more devastating consequences [21, 30, 35].

Potential mitigation strategies include not using sensitive features during model training and introducing appropriate regularisation techniques to combat the bias. For instance, in the *adult* dataset, the `sex` and `race` can be excluded from the training set so that the model can learn from more generalisable features such as the age and the level of education of an individual. More recently, the trustworthy AI research field has also seen significant developments to address issues regarding safety and robustness, explainability, fairness and privacy in machine learning models [31].

4.3.2 Imbalanced examples (misc-balance). The *fraud* dataset contains anonymised information on credit card transactions for over 200, 000 European cardholders. The dataset also contains a binary `class` feature where fraudulent transactions are assigned a value of 1 and others a value of 0. As seen from Figure 5b, a model trained on this dataset to detect fraudulent transactions will not perform well as the dataset contains very few examples for the fraudulent transactions class.

When performing classification using supervised learning algorithms, the target feature can also contribute to bias in the model. A special case of the *misc-sensitive* smell is the presence of unbalanced examples for the classes in a dataset.

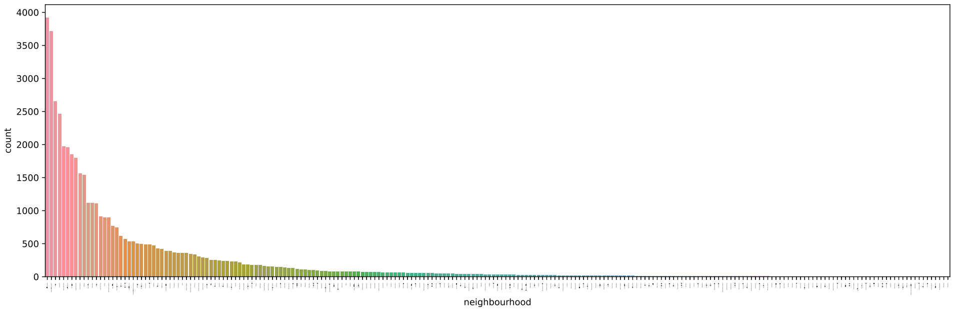
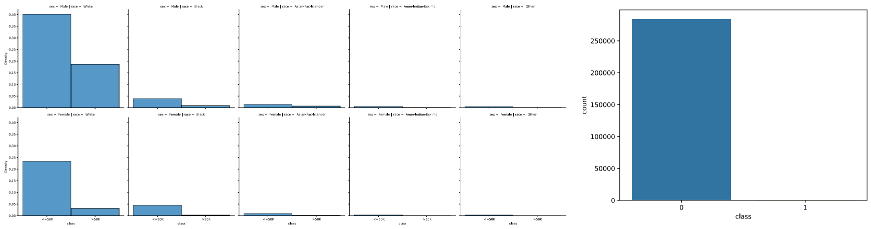


Figure 4: Histogram of `neighbourhoods` in *airbnb* dataset with high cardinality.



(a) Probability density plot of adult income based on their race (along x axis) and sex (along y axis).

(b) Histogram of `class` feature from *fraud* dataset.

Figure 5: Distribution of sensitive features from the *adult* and *fraud* datasets.

Common techniques for working with skewed datasets include using more robust performance metrics such as precision and recall, increasing the size of the training set or manipulating the training set such that the number of examples per class is equal.

4.3.3 Unknown unit of measure (misc-unit). The *breast-cancer* dataset contains several numerical features such as `radius`, `perimeter` & `area` of tumours. However the column names and documentation fail to mention the unit in which the features were measured.

Numerical data or data of the type `int` or `float` are the most common data types in structured datasets. Understanding the distribution and trends within numerical features constitutes the bulk of the data analysis stage. This helps in gaining a deep understanding of its characteristics which is vital to determine how a model will perform when trained on such a feature set. Consistency is of high importance when working with numerical features, and lack of a common unit of measure for all observations of a feature is an early indicator of problems during model training.

Lack of standardised data collection procedure, long durations of data collection and use of undocumented data sources can all lead to observations measured in different units. A mix of units can also lead to incorrect results from outlier detection and propagate to engineered features. Mean removal and variance scaling are common pre-processing techniques used to standardise the numerical features of a dataset prior to training a model. But such a transformation is unfruitful if the observations are not recorded with the same unit of measure.

4.4 Missing Value Smells

This section presents smells that arise from presence of missing values in datasets. This study identified 13 occurrences of smells from this group.

4.4.1 Nulltype missing values (miss-null). A common problem observed in the analysed datasets is the absence of data which can be an early indicator of potential problems in downstream processes. We noticed certain abnormalities while analysing the descriptive statistics of the *permit, bitcoin and covid-vaccine* datasets. Subsequently, the missing data check revealed that these datasets had large quantities of missing values. For instance, the *permit* dataset contains several datetime features where 50% of the data is missing; 25% of the data in the *bitcoin* dataset; and 50% of the data in the numerical features of the *covid-vaccine* dataset are missing.

When a small portion of the data is missing or if sufficient number of examples for each class is available, missing data can be omitted prior to further analysis [3]. Such a strategy ceases to be an option in high-stakes domains where data is limited and highly unbalanced to begin with. In such datasets, dropping missing data amplifies the imbalance and leads to insufficient data for training machine learning models [29, 42]. Missing values are typically ignored by data analysis tools while performing statistical computations (such as descriptive statistics), leading to inaccurate and biased conclusions. Missing values can also lower the performance of machine learning models due to underrepresented groups in the dataset [33, 39]. The problem gets worse as we scale to larger datasets which do not fit in the memory of a single computer and require a more distributed approach for storage and performing transformations.

Such cases require further effort from data scientists to impute the missing values. Imputation of missing data is a vast research field in itself with techniques ranging from simple statistical techniques such as mean, median and linear regression, to using machine learning models that predict the missing value [3, 40, 51].

4.4.2 Special missing values (miss-sp-val). The missing values in the *adult* dataset are represented with the question mark character. Although the null type (`null` or `nil`) is the most common data type used to represent missing values, sometimes special string characters and keywords such as `'?', 'nil', 'null'` are also used. *Dummy encoding* is another popular technique where a unique numerical value (such as `-9999` or `-6666`) which is unlikely to be observed in real-life is used. Using special characters, keywords and numbers to represent missing values—especially when they are undocumented—are a smell for problems in downstream stages.

Data analysis tools often contain built-in functionality to check for presence of missing values by detecting null data types in the dataset. Unless otherwise documented, using special characters or numbers impedes the ability of data scientists and data analysis tools to detect missing values accurately. We can indirectly discover string type missing values when performing statistical computations. In such a case, the data analysis tool will fail to perform a numerical operation on data represented as a string and raise an error. Usage of undocumented dummy encoding is however worse as the data analysis tool will continue to operate. This adds to technical debt and may result in incorrect statistical conclusions or catastrophic failure in downstream stages. The data must be manually analysed to identify the special character or number used to represent missing values, wasting time and effort that could have been used to perform other productive tasks.

Using null data types to represent missing values during data collection phase is regarded as good practise. Using special characters, keywords or dummy encoding for missing values must be documented to reduce technical debt and aid future practitioners.

4.4.3 Binary missing values (miss-bin). While analysing the *permit* dataset, we identified two features (namely `structural_notification` and `tidf_compliance`) with 90% missing values. All non-missing values however comprised of the string `'Y'`, a common abbreviation for *'yes'*. This indicated that the missing values carried an implicit meaning of `'N'` or *'no'* and were not indicative of truly missing values.

In Section 4.4.1 we saw how presence of excessive missing data can be a smell for incorrect statistical observations and additional effort for data imputation. Close attention however must be paid to the distribution of the missing values within the dataset. Presence of high quantities of missing data primarily within a column—as opposed to being distributed across rows and columns—can be a smell that the data is not truly missing. The missing values in such cases may carry an implicit meaning of a negative binary response.

This can be validated further by observing the column header along with the non-missing values of the feature(s) in question. If the non-missing data is indicative of a positive response such as `'{t,T}rue' or '{y,Y}es'` then the missing data may indicate a negative response. It is common practise in software engineering to represent a negative response or result using a null type, such as `None` in Python and `null` in Java. The same however does not hold in data science as null types are commonly used to represent missing data.

If a data scientist fails to notice this implicit meaning, they may hastily drop the missing values or perform imputation. However in doing so, the original information carried by the dataset is altered leading to inaccurate results and conclusions.

4.5 String Value Smells

This section presents smells that arise from the presence of string type data. This group of smells was least frequently observed with a total occurrence of 12.

4.5.1 Strings with special characters (str-sanitise). The missing values in the *adult* dataset are represented as `'?'`. However, the question marks also contain whitespaces, making their detection and subsequent imputation slightly more tedious. Another example can be found in the *abalone* dataset. This dataset contains the *sex* categorical feature where the value can be one of `'M', 'F' or 'I'`. However due to presence of whitespaces, the data analysis tools may consider values such as `' M', 'F ', ' I '` valid and distinct from one another.

The presence of leading and trailing whitespaces and special characters such as punctuation marks in structured data is a smell for potential problems in the data analysis stage.

Categorical features are often represented as strings during the data analysis phase and converted to a numerical representation prior to model training. The presence of whitespace and special characters in categorical features can confuse data analysis tools and lead to false results. The problem is easy to rectify in the *abalone* dataset since the set of correct values is 3. However things become more challenging for categorical features whose set of values are larger. For instance, the *airbnb* dataset contains the *neighbourhood* feature containing names of neighbourhoods in the United States. The feature contains over 200 valid values and presence of redundant whitespace and special characters can make the data cleaning process tedious and time consuming.

Handling presence of special characters in string features requires a case-by-case analysis and solution. But this smell flags the need to always check and remove leading and trailing whitespaces from string features. This is a common task performed by data scientists and popular data wrangling tools provide built in solutions.

4.5.2 Numerical features as string (str-num). The *playstore* dataset contains data for apps on the Google Playstore. The dataset contains the `current_ver` and `android_ver` features which represent the current version of the app and the supported android version respectively. The data in these columns are in the format of release versions such as `1.1.9`, which denotes the major, minor and patch versions of the latest release. Although the information is represented as string, we can extract 3 separate numerical features here which can provide valuable insights.

String features may sometimes contain numerical information embedded within itself. The smell here is the presence of features whose name indicates numerical type data, but the data analysis tool interprets the type as string.

Generally speaking, machine learning models tend to perform better when trained with more data. Extracting valuable numerical information from such features can therefore be beneficial for model training.

4.5.3 Strings in human-friendly formats (str-human). The *netflix* dataset contains information regarding content on the popular streaming service. The dataset contains the `duration` column which depicts the length of a particular movie or TV show as depicted in Table 5. In the case of movies, the data is clearly numerical in nature (duration in minutes) but represented as a string format that humans can easily comprehend (the *str-num* smell discussed earlier in Section 4.5.2 also applies here). The data for TV shows deviates from this format as the duration is represented as the number of seasons of the TV show. Although this format is also comprehensible to humans, converting them to a useful numerical representation however comes with several challenges.

Numerical information being represented in a human-friendly format is a smell for potential problems during the data analysis stage. This smell can be considered a subset of the *str-num* smell discussed earlier in Section 4.5.2.

Machine learning models generally perform better when trained with standardised and uniform data (also see Section 4.3.3). In this case, the duration should be represented in minutes for all examples. Converting the duration for TV shows from seasons to minutes can be difficult since the duration of each episode and the number of episodes in a season may vary amongst TV shows. We may wish to impute using average values however that requires domain knowledge or further investigation to be carried out by the data scientist.

Table 5: Excerpt from the *netflix* dataset showing the *str-human* data smell.

	type	duration
0	Movie	90 min
1	TV Show	2 Seasons
2	TV Show	1 Season
3	TV Show	1 Season
4	TV Show	2 Seasons
5	TV Show	1 Season
6	Movie	91 min
7	Movie	125 min
8	TV Show	9 Seasons
9	Movie	104 min

5 DISCUSSION

This section presents the key observations made in this study.

5.1 Documentation

This study identified several instances where a lack of proper documentation was felt. The *heart* dataset contains several cryptic column headers such as `cp, trestbps, & fbs`. This makes it difficult to understand what information the column provides without prior domain knowledge or further investigation. In the same dataset, the `sex` column is label encoded (ie. *male* and *female* are represented numerically). However without documentation we cannot ascertain the gender associated with the numerical values.

Improper documentation makes it difficult to understand the idiosyncrasies of a dataset such as determining if missing values are represented with special characters or if they carry an implicit meaning (see Section 4.4). Developing a deep understanding of the data is a fundamental step towards any data-centric work. Documentation can help in this process by providing useful metadata & context and help practitioners (re)familiarise themselves with the dataset. Our observations show the need for tools that aid machine learning practitioners with documenting their work. This is also corroborated in prior studies which show that machine learning practitioners spend considerable amount of time documenting their work as existing machine learning pipelines, models and datasets lack proper documentation [20, 23, 26, 42].

In line with recommendations made by *Hutchinson et al.* and *Sambasivan et al.*, good data documentation should include (but is not limited to) information regarding the data source and the data collection procedure, changes that may have already been made to the dataset along with the rationale behind the change, expected schema of the columns, meaningful column headers, presence of missing values & duplicate rows, correlation amongst features and descriptive statistics. Providing such documentation can significantly improve productivity of data scientists and reduce data understanding and development time [23, 42].

5.2 Technical debt

The analysis revealed technical debt in the datasets due to lack of best practices and standardised procedures in upstream processes. Undocumented data practices and transformations (Section 5.1), presence of missing values (Section 4.4) & redundant columns (Section 4.1), datasets with sensitive & imbalanced columns (Section 4.3) and data in human-friendly strings (Section 4.5.3), all lead to accumulation of technical debt in downstream stages.

Due to their highly tangled and experimental nature, machine learning systems are prone to rapid accumulation of technical debt [4, 5, 8, 46]. Although a holistic view is recommended for monitoring machine learning systems, it is often difficult to do so due to their complexity. Data smells however can help detect problems during the early stages of the machine learning lifecycle when the complexity is relatively low and fixes are easier and cheaper to implement. As with traditional software systems, accumulation of technical debt is inevitable. However early detection can improve effort estimation and help deliver projects on time with lower financial costs [18].

5.3 Data validation

Data validation tools provide an abstraction over common tests performed by data scientists when working with data. Analogous to how regression testing is done when introducing changes to a codebase, data validation ensures that the new data conforms to certain expectations when fed into a machine learning system [11]. But writing validation rules still requires data scientists to understand the data first. This may come naturally to seasoned practitioners, but is non-trivial for inexperienced practitioners [9].

As opposed to traditional rule-based software, machine learning models derive the rules automatically from the data. This reduces the level of human involvement in such systems, allowing for higher degrees of automation. Automation however comes at the cost of reduced transparency as minuscule changes to the input data can cause drastic changes in the trained model. Data validation and linting tools can automatically validate the data in terms of correctness, consistency, completeness and statistical properties. However they lack the ability to validate dimensions such as fairness and robustness which are critical in machine learning [7].

We believe that data smells can aid practitioners during the early stages of data analysis when human involvement is necessary. As seen in Section 4.3.1 and 4.2.1, data smells can aid practitioners to catch data quality issues that lead to biased and unfair predictions in their models. The smells also help fix other data quality issues that results in a more robust dataset.

5.4 Data Efficiency

The availability of highly resilient and cheap hardware commodity due to cloud-computing has enabled leaping advancements in AI. Neural networks have consistently evolved in complexity and size, starting with Imagenet in 2009, Resnet in 2015 and more recently GPT-3 in 2020 which consists of 175 billion parameters [10]. Complex models are data hungry, requiring training data in the scale of Terabytes. In this era of big data and high performance computing, presence of seemingly minor data smells can lead to wasted training cycles and cost millions [14]. Engineering efficient datasets become crucial in such circumstances and circumvent the use of overly complex machine learning models. Data quality plays a key role in delivering efficient and maintainable models with a smaller carbon footprint.

6 THREATS TO VALIDITY

This study opted for a *shallow* analysis of the datasets. The analysis phase consisted of a specific set of steps that were carried out (as outlined in Section 3) for each dataset. This was a deliberate decision, made to easily scale and reproduce the analysis steps across a large sample of datasets. We recognise that a *deeper* analysis of each dataset may reveal further smells. For instance, the current analysis excluded *outlier detection* or fitting machine learning models to the dataset. However we strongly believe that the generalisability of smells reduces as we increase the depth of the analysis. That is, the smells would only be valid within the context of the problem domain.

The smells discovered by this study are linked to the version of the data used for the analysis. For instance, *Kaggle* contains datasets from the *UCI Machine Learning Data Repository (UCI)*[8]. However the version of data found on Kaggle is different from the original source on UCI. Similarly, it is also unclear if the version of data hosted on UCI is the ground truth or was derived from somewhere else. While we recognise this issue, it is also unfortunately the nature of all data science problems. Table 1 provides a list of all datasets that were used in this analysis along with their version at the time it was downloaded in hopes to increase the reproducibility of the results.

The current analysis does not extend to quantifying impact of smells. For instance, we do not know if and to what extent the *unknown unit of measure (misc-unit)* smell affects the model's performance on a test set (see Section 4.3.3). Such a task requires collection of a sufficiently large sample of datasets that contain the smell in question. The validation process involves performing a supervised learning task which is a highly experimental and time consuming process. The effort and time increases when we scale the validation across multiple smells. Therefore such a time consuming task was beyond the scope of this study.

Data smells are subject to the interpretation of the data scientist, the team or the organisation performing the analysis. This problem of subjectiveness is present in code smells as well. Not all long methods are bad and god classes still exist in open source repositories. To reduce the possibility of subjective bias in our results, the smells were reviewed by the second author prior to including them in the catalogue.

7 CONCLUSION

Code smells are frequently used by software engineers to identify potential bugs, sources of technical debt and weak design choices. Code smells in the context of traditional software have existed for over three decades and have been extensively studied by the software engineering research community. With the growing popularity of AI and its adoption in high-stakes domains where a data-centric approach is adopted, data smells are seen as a much needed aid to machine learning practitioners. This study examined 25 public datasets and identified 14 recurrent data quality issues—coined as data smells—that can lead to problems when training machine learning models. Our results indicate a need for better data documentation, and accumulation of technical debt due to lack of standardised practices in upstream stages of machine learning pipelines.

We consider our collection of data smells and the analysis of their prevalence a first step towards aiding data scientists in the initial stages of data analysis where human involvement is necessary. We hope that our work raises awareness amongst practitioners to write better documentation for their datasets and follow best practises during data collection to minimise technical debt in upstream stages. As a next step, we aim to grow the data smells catalogue by analysing more datasets. Furthermore, we wish to remove the constraints introduced by *IC2* (see Section 3 and Table 2) and include datasets larger than 1GB in size.

[8] https://archive.ics.uci.edu/ml/datasets.php

REFERENCES

[1] 2018. Mortgage algorithms perpetuate racial bias in lending, study finds. https://news.berkeley.edu/story_jump/mortgage-algorithms-perpetuate-racial-bias-in-lending-study-finds/ Accessed on [2022-01-11 Tue].

[2] Taher Al-Shehari and Rakan A Alsowail. 2021. An Insider Data Leakage Detection Using One-Hot Encoding, Synthetic Minority Oversampling and Machine Learning Techniques. *Entropy* 23, 10 (2021), 1258.

[3] Tahani Aljuaid and Sreela Sasi. 2016. Proper imputation techniques for missing values in data sets. In *2016 International Conference on Data Science and Engineering (ICDSE)*. IEEE, 1–5.

[4] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 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, 291–300.

[5] Anders Arpteg, Björn Brinne, Luka Crnkovic-Friis, and Jan Bosch. 2018. Software engineering challenges of deep learning. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 50–59.

[6] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and Dave Binkley. 2015. Are test smells really harmful? an empirical study. *Empirical Software Engineering* 20, 4 (2015), 1052–1094.

[7] Felix Biessmann, Jacek Golebiowski, Tammo Rukat, Dustin Lange, and Philipp Schmidt. 2021. Automated Data Validation in Machine Learning Systems. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering.[Google Scholar]* (2021).

[8] Jan Bosch, Helena Holmström Olsson, and Ivica Crnkovic. 2021. Engineering AI systems: A research agenda. In *Artificial Intelligence Paradigms for Smart Cyber-Physical Systems*. IGI Global, 1–19.

[9] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Whang, and Martin Zinkevich. 2019. Data Validation for Machine Learning.. In *MLSys*.

[10] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[11] Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. 2016. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 international conference on management of data*. 2201–2206.

[12] Kate Crawford. 2021. *The Atlas of AI*. Yale University Press.

[13] Francisco Gonçalves de Almeida Filho, Antônio Diogo Forte Martins, Tiago da Silva Vinuto, José Maria Monteiro, Ítalo Pereira de Sousa, Javam de Castro Machado, and Lincoln Souza Rocha. 2019. Prevalence of bad smells in PL/SQL projects. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 116–121.

[14] Payal Dhar. 2020. The carbon impact of artificial intelligence. *Nature Machine Intelligence* 2, 8 (2020), 423–425.

[15] Emre Doğan and Eray Tüzün. 2022. Towards a taxonomy of code review smells. *Information and Software Technology* 142 (2022), 106737.

[16] Martin Fowler. 2006. CodeSmell. https://martinfowler.com/bliki/CodeSmell.html Accessed on [2022-01-04 Tue].

[17] Martin Fowler. 2018. *Refactoring: improving the design of existing code*. Addison-Wesley Professional.

[18] Yuepu Guo, Rodrigo Oliveira Spínola, and Carolyn Seaman. 2016. Exploring the costs of technical debt management–a case study. *Empirical Software Engineering* 21, 1 (2016), 159–182.

[19] Refactoring Guru. [n.d.]. Catalog of Refactoring. https://refactoring.guru/refactoring/catalog Accessed on [2022-01-04 Tue].

[20] Mark Haakman, Luís Cruz, Hennie Huijgens, and Arie van Deursen. 2021. AI lifecycle models need to be revised. *Empirical Software Engineering* 26, 5 (2021), 1–29.

[21] Will Douglas Heaven. 2020. Predictive policing algorithms are racist. They need to be dismantled. https://www.technologyreview.com/2020/07/17/1005396/predictive-policing-algorithms-racist-dismantled-machine-learning-bias-criminal-justice/ Accessed on [2022-01-11 Tue].

[22] Joseph M Hellerstein. 2008. Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)* 25 (2008).

[23] Ben Hutchinson, Andrew Smart, Alex Hanna, Emily Denton, Christina Greer, Oddur Kjartansson, Parker Barnes, and Margaret Mitchell. 2021. Towards accountability for machine learning datasets: Practices from software engineering and infrastructure. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. 560–575.

[24] Nick Hynes, D Sculley, and Michael Terry. 2017. The data linter: Lightweight, automated sanity checking for ml data sets. In *NIPS MLSys Workshop*.

[25] IBM. [n.d.]. IBM SPSS Modeler CRISP-DM Guide. https://www.ibm.com/docs/en/spss-modeler/SaaS?topic=guide-data-understanding Accessed on [2022-03-15 Tue].

[26] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2017. Data scientists in software teams: State of the art and challenges. *IEEE Transactions on Software Engineering* 44, 11 (2017), 1024–1038.

[27] Sanjay Krishnan, Michael J Franklin, Ken Goldberg, and Eugene Wu. 2017. Boostclean: Automated error detection and repair for machine learning. *arXiv preprint arXiv:1711.01299* (2017).

[28] Sanjay Krishnan, Daniel Haas, Michael J Franklin, and Eugene Wu. 2016. Towards reliable interactive data cleaning: A user survey and recommendations. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. 1–5.

[29] Meghana Kshirsagar, Caleb Robinson, Siyu Yang, Shahrzad Gholami, Ivan Klyuzhin, Sumit Mukherjee, Md Nasir, Anthony Ortiz, Felipe Oviedo, Darren Tanner, et al. 2021. Becoming Good at AI for Good. *AIES '21: Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society* (2021), 664–673.

[30] Brianna Lifshitz. 2021. Racism is systematic in artificial intelligence systems, too. https://georgetownsecuritystudiesreview.org/2021/05/06/racism-is-systemic-in-artificial-intelligence-systems-too/ Accessed on [2022-01-11 Tue].

[31] Haochen Liu, Yiqi Wang, Wenqi Fan, Xiaorui Liu, Yaxin Li, Shaili Jain, Yunhao Liu, Anil K Jain, and Jiliang Tang. 2021. Trustworthy ai: A computational perspective. *arXiv preprint arXiv:2107.06641* (2021).

[32] Lucy Ellen Lwakatare, Ellinor Rånge, Ivica Crnkovic, and Jan Bosch. 2021. On the experiences of adopting automated data validation in an industrial machine learning project. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 248–257.

[33] Benjamin Marlin. 2008. *Missing data problems in machine learning*. Ph.D. Dissertation.

[34] Fernando Martínez-Plumed, Lidia Contreras-Ochando, Cesar Ferri, José Hernández Orallo, Meelis Kull, Nicolas Lachiche, Maréa José Ramírez Quintana, and Peter A Flach. 2019. CRISP-DM twenty years later: From data mining processes to data science trajectories. *IEEE Transactions on Knowledge and Data Engineering* (2019).

[35] Natalia Mesa. 2021. Can the criminal justice system's artificial intelligence ever be truly fair? https://massivesci.com/articles/machine-learning-compas-racism-policing-fairness/ Accessed on [2022-01-11 Tue].

[36] Jennifer Miller. 2020. Is an Algorithm Less Racist Than a Loan Officer? https://www.nytimes.com/2020/09/18/business/digital-mortgages.html Accessed on [2022-01-11 Tue].

[37] Biruk Asmare Muse, Mohammad Masudur Rahman, Csaba Nagy, Anthony Cleve, Foutse Khomh, and Giuliano Antoniol. 2020. On the prevalence, impact, and evolution of SQL code smells in data-intensive systems. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 327–338.

[38] Andrew Ng. 2021. A Chat with Andrew on MLOps: From Model-centric to Data-centric AI. https://youtu.be/06-AZXmwHjo Accessed on [2022-01-17 Mon].

[39] Heru Nugroho and Kridanto Surendro. 2019. Missing Data Problem in Predictive Analytics. In *Proceedings of the 2019 8th International Conference on Software and Computer Applications*. 95–100.

[40] Asma Saleem, Khadim Hussain Asif, Ahmad Ali, Shahid Mahmood Awan, and Mohammed A Alghamdi. 2014. Pre-processing methods of data mining. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE, 451–456.

[41] Jeffrey S Saltz. 2021. CRISP-DM for Data Science: Strengths, Weaknesses and Potential Next Steps. In *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 2337–2344.

[42] Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. 2021. "Everyone wants to do the model work, not the data work": Data Cascades in High-Stakes AI. In *proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.

[43] Danilo Sato, Arif Wilder, and Christoph Windheuser. 2019. Continuous Delivery for Machine Learning. https://martinfowler.com/articles/cd4ml.html

[44] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann, and Andreas Grafberger. 2018. Automating large-scale data quality verification. *Proceedings of the VLDB Endowment* 11, 12 (2018), 1781–1794.

[45] Christoph Schröer, Felix Kruse, and Jorge Marx Gómez. 2021. A systematic literature review on applying CRISP-DM process model. *Procedia Computer Science* 181 (2021), 526–534.

[46] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. *Advances in neural information processing systems* 28 (2015), 2503–2511.

[47] Tushar Sharma, Marios Fragkoulis, Stamatia Rizou, Magiel Bruntink, and Diomidis Spinellis. 2018. Smelly relations: measuring and understanding database schema quality. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. 55–64.

[48] Davide Spadini, Fabio Palomba, Andy Zaidman, Magiel Bruntink, and Alberto Bacchelli. 2018. On the relation of test smells to software code quality. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 1–12.

[49] Michele Tufano, Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrea De Lucia, and Denys Poshyvanyk. 2016. An empirical investigation into the nature of test smells. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. 4–15.

[50] Erdem Tuna, Vladimir Kovalenko, and Eray Tüzün. 2022. Bug Tracking Process Smells In Practice. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. IEEE.

[51] Bhekisipho Twala, Michelle Cartwright, and Martin Shepperd. 2005. Comparison of various methods for handling incomplete data in software engineering databases. In *2005 International Symposium on Empirical Software Engineering, 2005*. IEEE, 10–pp.

[52] Zhiyuan Wan, Xin Xia, David Lo, and Gail C Murphy. 2019. How does machine learning change software development practices? *IEEE Transactions on Software Engineering* (2019).

[53] Mark Weber, Mikhail Yurochkin, Sherif Botros, and Vanio Markov. 2020. Black Loans Matter: Fighting Bias for AI Fairness in Lending.

https://mitibmwatsonailab.mit.edu/research/blog/black-loans-matter-fighting-bias-for-ai-fairness-in-lending/ Accessed on [2022-01-11 Tue].

[54] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. 2020. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering* (2020).

[55] Marc-André Zöller and Marco F Huber. 2021. Benchmark and survey of automated machine learning frameworks. *Journal of Artificial Intelligence Research* 70 (2021), 409–472.