2021 IEEE 37th International Conference on Data Engineering (ICDE) | 978-1-7281-9184-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICDE51399.2021.00009

CleanML: A Study for Evaluating the Impact of Data Cleaning on ML Classification Tasks

Peng Li†, Xi Rao‡, Jennifer Blase†, Yue Zhang†, Xu Chu†, Ce Zhang‡
†*Georgia Institute of Technology*, ‡*ETH Zurich*
†{pengli, jblase, yzhang3271, xu.chu}@gatech.edu, ‡{rao, ce.zhang}@inf.ethz.ch

***Abstract*—Data quality affects machine learning (ML) model performances, and data scientists spend considerable amount of time on data cleaning before model training. However, to date, there does not exist a rigorous study on how exactly cleaning affects ML — ML community usually focuses on developing ML algorithms that are robust to some particular noise types of certain distributions, while database (DB) community has been mostly studying the problem of data cleaning alone without considering how data is consumed by downstream ML analytics.**

We propose a CleanML study that systematically investigates the impact of data cleaning on ML classification tasks. The open-source and extensible CleanML study currently includes 14 real-world datasets with real errors, five common error types, seven different ML models, and multiple cleaning algorithms for each error type (including both commonly used algorithms in practice as well as state-of-the-art solutions in academic literature). We control the randomness in ML experiments using statistical hypothesis testing, and we also control false discovery rate in our experiments using the Benjamini-Yekutieli (BY) procedure. We analyze the results in a systematic way to derive many interesting and nontrivial observations. We also put forward multiple research directions for researchers.

I. Introduction

The quality of machine learning (ML) applications is only as good as the quality of the data it trained on, and data cleaning has been the cornerstone of building high-quality ML models. Not surprisingly, both ML and database (DB) communities have been working on problems associated with dirty data:

- **ML community** has been focusing on understanding the impact of noises on ML models without actually performing data cleaning. On the one hand, many ML models are robust to small amounts of random noises — there has been research showing that the noise introduced during the training process (e.g., via asynchronous communication and lossy compression) can have negligible effect in accuracy, both empirically and theoretically [5, 17, 35, 43, 52, 53]. On the other hand, ML models can also be sensitive to other types of noises, especially those non-white noises that are in the input data [33] and labels [21]. Instead of performing data cleaning, the ML community has mostly been focusing on designing ML algorithms that are *robust* to noises of certain distributions, such as noise-robust decision trees [41], the use of regularization for improving robustness [48], and model bagging to reduce the variability of model performances caused by dirty data [30].
- **DB community** has been mostly focusing on understanding the fundamental process of data cleaning without considering its impact on ML models. Data cleaning usually consist of two phases: *error detection*, where various errors are identified and possibly validated by experts; and *error repair*, where updates to the database are applied (or suggested to human experts) to make the data cleaner. Many techniques have been proposed for detection, for example, by designing integrity constraints to capture data inconsistencies [14], by using statistical techniques to detect outliers [26], and by building ML models to detect duplicates [19]. Various techniques have also been proposed for repairing, for example, by finding the minimal set of updates to resolve violations [15], by performing data transformations [24], by consulting external knowledge bases [16], and by using probabilistic graphical models to reason about errors holistically [44].

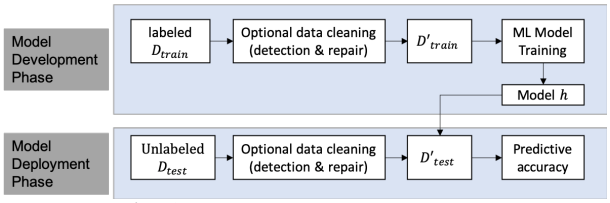

Fig. 1. Typical ML workflow with data cleaning.

(Goal of This Work.) In real applications, these two angles are less segregated. Indeed, data cleaning is often seen as a crucial data preparation step either before training an ML model on a labeled training set in the model development phase, or before making predictions on an unlabeled test set in the model deployment phase [40] (Figure 1). It is reported that data scientists reportedly spend up to 80% of their time performing various data cleaning activities [1].

However, limited prior work exists in studying the impact of data cleaning for downstream ML model performance, and they tend to focus on some specific error types (e.g., domain value errors in BoostClean [32]), specific cleaning methods (e.g., human oracle for cleaning in ActiveClean [33]), and/or specific ML models (e.g., convex models trained using stochastic gradient descent in ActiveClean [33] and a weighted ensemble model in BoostClean [32]).

The goal of this paper is not to propose a new approach of data cleaning for ML. Instead, our goal is to (1) conduct a first systematic empirical study on the impact of data cleaning on downstream ML classification models, for different error types, cleaning methods, and ML models; (2) given our empirical findings, provide a starting point for future research to advance the field of cleaning for ML.

978-1-7281-9184-3/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDE51399.2021.00009

(Challenges) Conducting such a systematic study is not a trivial task with the following challenges:

- **Comprehensive Study Scope.** As shown in Figure 1, the accuracy of an ML model depends on the dataset, the ML model, whether data cleaning is applied, where data cleaning is applied (on training data or on test data), and what cleaning algorithm is used. Studying a particular aspect of the workflow (e.g., logistic regression as [47]) is not enough to provide a qualitative assessment on the general impact of data cleaning. We thus need a principled way to organize different experiments and analyze results systematically.
- **Realistic Error Patterns.** Most of the current data cleaning work in the DB community uses synthetically injected errors (and hence the ground-truth clean data is known) to evaluate data cleaning algorithms [7]. Taking a standard ML dataset with simulated data fallacies (e.g., by randomly removing values to mimic missing values) might under/over-estimate the impact of data cleaning on ML. For our study to reflect the real-world impact of data cleaning on ML, we have to work with real-world datasets that contain realistic errors, for which we usually do not have ground-truth clean versions.
- **Statistically Significant Results and Controlling False Discoveries.** ML models are inherently probabilistic, for example, a different train/test split on the same dataset might produce entirely different results. Therefore, ensuring the statistically significance of our findings presents a major challenge. This is further complicated by the fact that we have many different datasets, ML models, and data cleaning choices to cover, which will likely produce false discoveries, known as the multiple hypothesis testing problem [46, 54].

(Main Contributions of CleanML.) We make the following contributions in our proposed CleanML study.

- **CleanML Scope of Study.** We focus on *five types of errors*, including outliers, duplicates, inconsistencies, mislabels and missing values. These errors are prevalent in the real-world datasets and frequently considered in research. We select *seven classification algorithms*, including Logistic Regression, Decision Tree, Random Forest, Adaboost, XGBoost, k-Nearest Neighbors (KNN) and Naive Bayes. These are classical and competitive classification models commonly used on classification tasks on structured datasets. We collected *14 real-world datasets* containing different types of errors, and devised meaningful classification tasks on the datasets. We organize all experimental results in a relational database named CleanML, whose schema we describe in Section III.
- **Obtaining Cleaned Versions of Dirty Datasets.** As we do not have ground-truth versions of the dirty datasets, for each error type, we selected various automatic data cleaning algorithms, including both simple data cleaning solutions commonly used in practice (e.g., mean imputation), as well as state-of-the-art data cleaning solutions proposed in academic literature (e.g., probabilistic cleaning solution HoloClean [44] and unsupervised duplicate detection solution ZeroER [51]). We also include a study comparing human cleaning with automatic cleaning.
- **Controlling Randomness and False Discoveries.** We describe the steps taken to populate the CleanML database instance in Section IV, where we controlled randomness by using multiple train/test splits to conduct a statistical hypothesis testing procedure, the paired sample t-test [36]. We also leveraged the Benjamini-Yekutieli (BY) procedure [22] to control false discovery rate.
- **Empirical Findings and Future Research Directions.** We obtain many interesting and non-trivial empirical findings by systematically issuing various queries against the CleanML relational database. We present a detailed analysis for each error type in Section V. We summarize our findings in Section VI. We also put forward multiple directions for future research in the area of cleaning for ML in Section VIII.
- **Extensible Open-source Code.** We made the code and datasets publicly available for reproducibility [1]. In addition, our CleanML study can be easily extended by adding new datasets, error types, cleaning algorithms, or ML models — the code for running experiments and for performing result analysis can be reused without modification.

II. Related Work

Data Cleaning. Despite many years of research, the dirty data problem remains challenging [13]. Recent study shows that even the combination of current error detection techniques can still miss many errors in real-world datasets [2]. We refer readers to various surveys and tutorials on the broad topic of data cleaning [20, 26, 28, 42]. The recent efforts on data cleaning focus on leveraging advanced ML techniques, including the HoloDetect system that uses few-shot learning and data augmentation for error detection [25], the HoloClean system that uses probabilistic graph models for inferring the most likely values [44], deep-learning based methods for duplicate detection [18, 37], and the ZeroER that uses generative model for detecting duplicates with zero labeled examples [51].

Analytics-Driven Cleaning. As data cleaning itself is expensive and hard to reach ground truth, the DB community is starting to work on analytics-driven cleaning methods. SampleClean [50] targets the problem of answering SQL aggregate queries when the input data is dirty by cleaning a sample of the dirty dataset, and at the same time, providing statistical guarantees on the query results. ActiveClean [33] is an example of cleaning data for convex ML models that are trained using gradient descent. The key insight of ActiveClean is that convex loss models (e.g., linear regression) can be trained and cleaned simultaneously using mini-batch gradient descent. ActiveClean assumes that the cleaning is performed by a human oracle while CleanML evaluates automatic cleaning algorithms. BoostClean [32] automatically selects from a predefined space of cleaning methods using a hold-out validation set via statistical boosting. BoostClean only considered *domain value violations* when an attribute value is outside of its value domain and only tested random forest model, while CleanML considers five error types and seven ML models.

[1] https://chu-data-lab.github.io/CleanML/

III. CleanML Database Schema

As shown in Figure 1, there are multiple factors contributing to the impact of data cleaning on ML classification tasks. To effectively organize the study, we propose to use a relational database (named the CleanML database) to store the results of different experiments. We first discuss the three relations in the CleanML database in Section III-A. We then present the domain of each attribute in the relations from Sections III-B to III-E, which together define the scope of our study.

A. The Three Relations

TABLE 1. CleanML Schema. Keys are underlined.

R1 (Vanilla)

Dataset	Error Type	Detection	Repair	ML Model	Scenario	Flag

R2 (With Model Selection)

Dataset	Error Type	Detection	Repair	Scenario	Flag

R3 (With Model Selection and Cleaning Method Selection)

Dataset	Error Type	Scenario	Flag

The CleanML relational schema consists of three relations, as shown in Table 1. The primary keys of all three relations are underlined, respectively. We first introduce the attributes, and then highlight the differences between these three relations.

- **Dataset Attribute.** The first attribute is *dataset*, which specifies a dataset under study. Each dataset has an associated ML task. Each dataset can have multiple types of errors. Instead of injecting synthetic errors into datasets, we use real-world datasets with real errors, and we apply various cleaning methods to detect and repair the errors. We list all the datasets we use in Section III-C.
- **Attributes for Data Cleaning.** The *error type* attribute specifies the error type under consideration. We include five most common types of dirtiness considered in the ML and DB communities: missing values, outliers, duplicates, inconsistencies and mislabels. For each error type, we consider the most commonly used cleaning methods in practice as well as the state-of-the-art cleaning algorithm in academic literature. Each cleaning method includes an error *detection* component and an error *repair* component. We discuss the cleaning methods considered in Section III-B. We not only look at single error cleaning, but also discuss mixed error cleaning in Section VII-A.
- **Attribute for ML Model.** The *ML model* specifies the ML algorithm used for classification. Different ML models may have different behaviors to different error types. We describe the chosen ML models in Section III-D.
- **Attribute for Cleaning Scenario.** The *scenario* attribute indicates whether cleaning is applied in the training set or the test set (c.f. Figure 1). In other words, it specifies whether we are evaluating the impact of data cleaning on ML in the model development phase or in the model deployment phase. We explain this further in Section III-E.
- **Flag Attribute.** The *flag* attribute summarizes the impact of data cleaning on ML for a specific setting under consideration, i.e., a particular combination for the key attributes. It has three possible values: "P (positive)", "N (negative)" or "S (insignificant)", indicating the cleaning has positive, negative, or insignificant impact on the ML performance, respectively.

Relation $R1$**.** $R1$ is the vanilla version of our CleanML relations. Its primary key is *dataset, error type, detection, repair, ML model, scenario*. Every tuple of $R1$ stores the result (flag) of a specific experiment: *how does cleaning some type of error using a detection method and a repair method affect a ML model for a given dataset?*

Relation $R2$**.** Given an ML classification task, ML developers often perform model selection to select the best model (e.g., by using cross-validation). Compared with $R1$, $R2$ eliminates the ML model attribute. In this case, we try different models during training and select the model that has the best validation accuracy (or F1 score if the dataset has imbalanced classes) as the model to be considered in an experiment in $R2$. Every tuple of $R2$ stores the result (flag) of a specific experiment with model selection: *how does cleaning some type of error using a detection method and a repair method affect the best ML model for a given dataset?* Note that we must create a new relation $R2$ for this question, instead of simply picking one tuple out of the tuples in $R1$ with the same dataset, error type, detection, repair, and scenario attribute values. This is because every tuple in $R1$ is actually an aggregation over different train/test splits in order to control randomness in the experiments (c.f., Section IV-B), and the best model for each train/test split might be different.

Relation $R3$. Compared with $R2$, $R3$ further eliminates the cleaning method (detection and repair) attributes. In this case, in addition to model selection, we also try different cleaning methods and select the one that results in the best validation accuracy (or F1 score if the dataset has imbalanced classes). This is essentially the cleaning algorithm selection strategy employed in BoostClean [32]. Every tuple of $R3$ stores the result (flag) of a specific experiment with model selection and cleaning method selection: *how does the best cleaning method affect the performance of the best model for a given dataset?* Again, we must create a new relation $R3$ for addressing this question, instead of picking one tuple from $R2$ with the same dataset, error type, and scenario attribute values. This is because every tuple in $R2$ is also an aggregation over different train/test splits (c.f., Section IV-B), and the best cleaning method for each split may be different.

B. Error Types and Automatic Cleaning Methods

We consider five error types prevalent in the real-world datasets, including missing values, outliers, duplicates, inconsistencies and mislabels. For each error type, we consider various automatic cleaning methods commonly used in practice as well as the state-of-the-art cleaning methods in academic literature. Note that all algorithms discussed in this section are automatic cleaning algorithms that require no or minimal human interactions. We will separately study human cleaning in Section VII-C. All automatic cleaning methods are described below and listed in Table 2.

1) Missing Values: Missing values occur when no value is stored for cells. Missing values can be naturally detected by finding empty or NaN (a commonly used placeholder) entries. We use the following methods to repair missing values:

TABLE 2. Automatic Cleaning Methods

Error Type	Detection Method	Repair Method
Missing Values	Empty Entries	Deletion
		Mean_Mode, Mean_Dummy Median_Mode, Median_Dummy Mode_Mode, Mode_Dummy
		HoloClean
Outliers	SD	Mean, Median, Mode
	IQR	
	IF	HoloClean
Duplicates	Key Collision	Deletion
	ZeroER	
Inconsistencies	OpenRefine	Merge
Mislabels	cleanlab	cleanlab

- **Deletion:** Delete records with missing values.
- **Six Ways of Simple Imputation:** For numerical missing values, we consider three types of imputation methods: mean, median and mode. For categorical missing values, we use two types of imputation methods: the mode (most frequent class) or a dummy variable named "missing". Therefore, we have six imputation methods. In Table 2, we denote each imputation method by the numerical imputation and categorical imputation (e.g. "Mean_Dummy" represents imputing numerical missing values by mean and imputing categorical missing values by dummy variables).
- **Probabilistic Inference (HoloClean):** HoloClean [44] is a state-of-the-art statistical inference engine to impute, clean, and enrich data. It leverages multiple signals (e.g., value correlations and any available reference data) holistically to build a probabilistic model for inferring what is the most likely value for a given cell.

2) Outliers: An outlier is an observation that is distant from others [4]. We detect numerical outliers as follows:

- **Standard Deviation Method (SD):** A value in a column is considered to be an outlier if it is n numbers of standard deviations away from the mean. We use $n = 3$.
- **Interquartile Range Method (IQR):** Let Q_1 and Q_3 be the 25th and the 75th percentiles of an attribute. Then, the interquartile range $IQR = Q_3 - Q_1$. A value is considered to be an outlier if it is outside the range of $[Q_1 - k \times IQR, Q_3 + k \times IQR]$. We use $k = 1.5$.
- **Isolation Forest Method (IF):** The isolation forest isolates observations by randomly selecting a feature and a split value of the selected feature. This partition can be represented by a tree structure and it produces noticeably shorter paths for outliers. We use the scikit-learn IsolationForest [39] and set the contamination parameter to be 0.01.

We repair outlying cells using simple imputation and HoloClean, which are exactly the same as those for repairing missing values, except that we only have three ways of imputation since we consider only numerical outliers.

3) Duplicates: Duplicates refer to the records that correspond to the identical real-world entity [19]. We consider a simple method commonly used in practice as well as a state-of-the-art unsupervised approach for detecting duplicated records:

- **Key Collision:** This method uses the key attributes that are supposed to be unique for every tuple in a dataset, and declares two records as duplicates if they have the same value on the key attributes.
- **Unsupervised ER (ZeroER):** While there are many supervised approaches for detecting duplicates, including recent deep learning based methods [18, 37], real-world datasets rarely come with labeled duplicates and non-duplicates. Hence, we choose a state-of-the-art unsupervised method, called ZeroER [51], that achieves comparable performance to supervised methods but requires zero labeled examples.

For a set of records that are deemed to be duplicates, we repair them by deleting all but one record in the set.

4) Inconsistencies: Inconsistencies occur when two cells in a column have different values, but should actually have the same value [28]. For example, both "CA" and "California" can appear together in a state column. We use a popular open-source tool for detecting inconsistencies:

- **OpenRefine:** *OpenRefine* [49] provides a *text facets clustering* function, which is able to find groups of different values that might be alternative representations of the same thing. For example, after clustering text facets on the company name attribute, "U.S. Bank" and "US Bank" will be clustered into the same group and can be identified as inconsistencies. We use this function to detect inconsistencies. Fixing inconsistencies can be done by simply merging all values in one cluster into the most frequent one.

5) Mislabels: Mislabels occur when an example is incorrectly labeled. As we only have one dataset (Clothing) with real mislabels, we perform synthetic mislabel injection on additional four datasets (c.f. Table 3) , following the strategies in [23]: (1) *uniform* class injection: flip 5% of the labels in each class; (2) *majority* class injection: flip 5% of the labels in the majority class; and (3) *minority* class injection: flip 5% of the labels in the minority class.

- ***cleanlab:*** We employ *cleanlab* [38] to automatically clean the mislabels and then run the ML models, because it is model agnostic and can be easily configurable with any downstream model. *cleanlab* implements confident learning with provable guarantees of exact noise estimation and label error finding.

C. Datasets

We collected 14 real-world datasets[2] with varying error types and error rates, as summarized in Table 3.

TABLE 3. Dataset and Error Types

Datasets	Error Types				
	Inconsistencies	Duplicates	Missing Values	Outliers	Mislabels
Citation		x			
EEG				x	x
Marketing			x		x
Movie	x	x			
Company	x				
Restaurant	x	x			
Sensor				x	
Titanic			x		x
Credit			x	x	
University	x				
USCensus			x		x
Airbnb		x	x	x	
BabyProduct			x		
Clothing					x

D. ML Models

We selected seven classical and competitive ML models commonly used in classification tasks on structured datasets,

[2]All datasets, their descriptions and error prevalence in each dataset can be found in https://github.com/chu-data-lab/CleanML/blob/master/DatasetDescriptions.pdf.

including Logistic Regression, k-Nearest Neighbors (KNN), Decision Tree, Random Forest, AdaBoost, Naive Bayes and XGBoost [12]. We use scikit-learn [39] for training models.

E. Scenarios

Data cleaning may be applied either in the model development phase on the training data, or in the model deployment phase on the test data (c.f. Figure 1). Depending on where data cleaning is applied, we can have four different performance metrics, as shown in Table 4: (1) *Case A* represents a model built using the original dirty training set and tested on the original dirty test set; (2) *Case B* represents a model built using the original dirty training set and tested on the cleaned test set; (3) *Case C* represents a model built using the cleaned training set and tested on the original dirty test set; and (4) *Case D* represents a model built using the cleaned training set and tested on the cleaned test set.

TABLE 4. Four different performance metrics on where data cleaning is performed

	Dirty Test Set	**Cleaned Test Set**
Dirty Training Set	*A*	*B*
Cleaned Training Set	*C*	*D*

TABLE 5. Four different performance metrics on how missing value is handled

	Test Set by Deletion	**Test Set by Imputation**
Training Set by Deletion	*A*	*B*
Training Set by Imputation	*C*	*D*

We aim to evaluate the impact of cleaning on ML model performance both in the model development phase and in the model deployment phase using these four metrics:

Model Development Scenario (BD). During the model development phase, ML developers would like to assess whether cleaning the training data can potentially improve the model performance on unseen test data. Therefore, we need to train two models: one trained on the original dirty training set and one trained on the cleaned version. To compare the two models on equal grounds, we need to evaluate them on the same test set. We can either compare the two models on the same dirty test set (i.e., compare *Case A* with *Case C*) or the same cleaned test set (i.e., compare *Case B* with *Case D*). We choose to compare *Case B* with *Case D* as it is common practice to ensure that the test set is cleaned for performance evaluations.

Model Deployment Scenario (CD). During the model deployment phase, a particular ML model has been trained and deployed into production to make predictions on incoming test data. In this phase, ML developers would like to know whether cleaning the test data can improve the performance of an already trained model. Hence, we can either compare *Case A* with *Case B*, or compare *Case C* with *Case D*. We choose to compare *Case C* with *Case D* because they use the cleaned training data, which often produces a better model.

Special Consideration for Missing Values. Missing values need special attention, as we cannot train a model or make predictions on a dataset with missing values. This means that *Cases A*, *B*, *C* in Table 4 are not available when dealing with missing values. To deal with this issue, we treat the dataset by deleting records with missing values as the "dirty" dataset, and a cleaned dataset is only achieved by filling in the missing values, as shown in Table 5. In addition, we can only consider the model development scenario (BD) for missing values since deleting records from test set (i.e., *Case A* and *Case C*) is usually not acceptable in practice — the missing values in test set have to be filled in to make predictions.

IV. CleanML Database Instance

TABLE 6. Example of Experiment Specifications

s_1

Dataset	**Error Type**	**Detection**	**Repair**	**ML Model**	**Scenario**
EEG	Outliers	IQR	Mean Imputation	Logistic Regression	BD

s_2

Dataset	**Error Type**	**Detection**	**Repair**	**Scenario**
EEG	Outliers	IQR	Mean Imputation	BD

s_3

Dataset	**Error Type**	**Scenario**
EEG	Outliers	BD

In this section, we show how to populate the CleanML database instance. We have presented all possible values for all attributes in the primary keys of $\{R1, R2, R3\}$. Each combination of value assignment for the key in $R \in \{R1, R2, R3\}$ defines a particular setting, which we call *experiment specification*. Our goal is to determine the value for the "flag" attribute for each experiment specification, i.e., what is the impact of cleaning on ML for a given experiment specification. Table 6 shows three example experiment specifications s_1, s_2 and s_3 in $R1$, $R2$ and $R3$, respectively.

A. Generating One Performance Metric Pair

Given an experiment specification s_1 in $R1$, we can generate a pair of performance metrics through following steps:

(1) **Splitting dataset.** We split the dataset in s_1 randomly into a training and a test set with a 70/30 ratio.

(2) **Performing data cleaning.** We clean the error type given in s_1 in both the training set and the test set, using the cleaning algorithm defined in s_1. To avoid any data leakage, all statistics necessary for data cleaning, such as mean, are computed only on the training set, and are used to clean both the training and the test set.

(3) **Training ML models.** If the scenario is BD in s_1, we train two ML models, one on the original dirty training set and one on cleaned training set. If the scenario is CD, we only need to train one ML model on the cleaned training set. We perform hyper-parameter tunings using standard random search and 5-fold cross validation.

(4) **Evaluating ML models.** If the scenario is BD in s_1, we evaluate the two ML models on the cleaned test set to get a pair of performance metrics. If the scenario is CD, we evaluate the one model on the dirty test set and the clean test set, respectively, to get a pair of performance metrics. We use the classification accuracy as the metric on all datasets, except for the class-imbalanced datasets (e.g. Credit) for which we use F1 score as the metric.

Given an experiment specification s_2 in $R2$ without a specific model, we perform the same four steps with the following modification for Step (3). We train all seven ML models (c.f. Section III-D) and select the model with the best validation accuracy from cross validation. The selected model is then used to obtain a pair of metrics in Step (4).

Given an experiment specification s_3 in $R3$ without a specific model and a specific cleaning method, we try all available cleaning methods (c.f. Section III-B) for the error

type given in s_3 to clean dataset at Step (2). We will select the cleaning method and the ML model that has the best validation accuracy in Step (3). The selected model and cleaning method is then used to obtain a pair of metrics in Step (4).

TABLE 7. s_1 Metric Pairs

Model	*Train on Dirty Training set*		*Train on Clean Training set*	
	Val Acc	**Clean Test Acc**	**Val Acc**	**Clean Test Acc**
LR	0.638	0.634	0.673	0.668
Metric Pair: (0.634, 0.668)				

TABLE 8. s_2 Metric Pairs

Model	*Train on Dirty Training set*		*Train on Clean Training set*	
	Val Acc	**Clean Test Acc**	**Val Acc**	**Clean Test Acc**
Adaboost	0.763	0.711	0.718	0.715
KNN	0.895	0.821	**0.948**	**0.956**
XGBoost	**0.932**	**0.862**	0.920	0.922
Metric Pair: (0.862, 0.956)				

TABLE 9. s_3 Metric Pairs

Detection	Repair	*Best Model on Clean Training Set*	*Best Model on Dirty Training Set*	*Best Model on Clean Training Set*
		Val Acc	**Clean Test Acc**	**Clean Test Acc**
SD	**Mean**	**0.959**	**0.937**	**0.969**
SD	Median	0.955	0.938	0.964
SD	Mode	0.955	0.937	0.964
...	...	...	...	...
Metric Pair: (0.937, 0.969)				

TABLE 10. Accuracy Evaluated on the Clean Test Set

Split Seed	1	2	3	4	...	19	20
B	0.632	0.631	0.634	0.638	...	0.629	0.632
D	0.657	0.674	0.668	0.676	...	0.669	0.668

Example 4.1: To generate *a metric pair for* s_1 in Table 6, we first split EEG into training/test sets. We detect outliers in the training set and test set using IQR detection and repair them with mean imputation. The quantiles used in detection and mean used in repair are computed on the training set. Since the scenario here is BD, we train two logistic regression models on the dirty training set and the cleaned training set, respectively. Finally, we evaluate the two models on the cleaned test set to obtain two test accuracy scores to form a metric pair, as shown in Table 7. To generate *a metric pair for* s_2, we train all seven ML models. As shown in Table 8, based on the validation accuracy, XGBoost is the best model trained on the dirty training set, and KNN is the best model on the cleaned training set. We then evaluate the two models on the same cleaned test set to get two accuracy scores that form a metric pair. To generate *a metric pair for* s_3, in addition to model selection, we use all available methods for cleaning outliers. As shown in Table 9, detecting outliers by SD and repairing by mean imputation is the best cleaning method, as it has the best validation accuracy among all cleaning methods. Hence, we use its metric pair as the metric pair for s_3.

B. Handling Randomness

While the procedure described in Section IV-A produces one pair of metrics given an experiment specification, it may produce an entirely different pair of metrics for a different train/test split. To handle ML randomness, we conduct the same procedure 20 times, each time with a different train/test split. Table 10 shows 20 pairs of metrics we obtained given s_1 in Table 6. Given the 20 metric pairs, one might be tempted to take the average and set the "flag" attribute based on the difference between the two metrics in the averaged metric pair: if the averaged $D > B$, flag is set to "P" (positive); if the averaged $D < B$, flag is set to "N" (negative); if the averaged $D = B$, flag is set to "S" (insignificant); This simple approach is problematic since the absolute difference between the two averaged metrics may not be significant enough.

We propose to follow a rigorous statistical significance testing procedure to determine the flag based on 20 metric pairs. Specifically, we use the *paired sample t-test* [36], which is commonly used to determine whether the mean difference between two sets of observations is zero, positive, or negative. For example, it can be used to determine whether taking a course has a positive impact on student academic performance by comparing test scores of 20 students before and after taking the course. As far as we know, existing applications of paired sample t-test are often used to determine a binary outcome (e.g., positive or non-positive). However, our application needs to produce a three-valued flag attribute. We describe our process to determine the flag attribute in the following.

Let μ^t be the mean difference of the performance metrics before and after data cleaning. We use three paired sample t-test with the following null and alternative hypotheses:

Hypothesis	Two-tailed t-test	Upper-tailed t-test	Lower-tailed t-test
Null	$H_0^t : \mu^t = 0$	$H_1^t : \mu^t \leq 0$	$H_2^t : \mu^t \geq 0$
Alternative	$H_a^t : \mu^t \neq 0$	$H_b^t : \mu^t > 0$	$H_c^t : \mu^t < 0$

We determine the flag attribute based on the three tests as follows, where p_0, p_1, p_2 denote the p-values of two-tailed t-test, upper-tailed t-test, and lower-tailed t-test respectively and α denotes the significant level.

(1) if $p_0 \geq \alpha$, Flag = "S".
(2) if $p_0 < \alpha$ and $p_1 < \alpha$, Flag = "P".
(3) if $p_0 < \alpha$ and $p_2 < \alpha$, Flag = "N".

The intricacy of conducting three tests together lies in the fact that, if the test statistics distribution is symmetric (e.g., Gaussian), the p-value in one of the one-tailed tests is exactly half of the p-value in the two-tailed test. Hence, a two-tailed test with significance implies that one of the one-tailed tests is significant; yet if the one-tailed test is significant, the two-tailed one is not necessarily significant. What is criticized often by people is to only report the significance by a one-tailed test because the two-tailed test is insignificant. However, we do not face this claim because we conduct three tests and only report the one-tailed test results if the two-tailed test is significant.

Example 4.2: For s_1, we run these three tests on 20 metric pairs (c.f. Table 10). The three p-values we obtain are: $p_0 = 3.82e^{-17}$, $p_1 = 1.91e^{-17}$, and $p_2 = 1$. α is set to 0.05. Hence, we determine the "flag" attribute for s_1 as "P".

C. Controlling False Discoveries

Given the comprehensive scope we have in our study, we have many hypothesis testing procedures to run, which can cause *false discoveries*, namely, many hypotheses are incorrectly declared as significant. This is commonly known as the *multiple hypothesis testing* problem in the statistics literature [46]. To see the effect of multiple testing, consider a case where there are 20 hypotheses to test and we set a significance level of $\alpha = 0.05$. The probability of observing at least one significant result due to chance is $1-(1-\alpha)^{20} \approx 0.64$. Thus, we have at least 64% chance of observing significant results within just 20 tests, even if all tests are actually insignificant.

With 3612, 516 and 168 hypotheses in our relations $R1$, $R2$ and $R3$ (3 $\times$ the number of unique assignments for key attributes), respectively, it is highly likely that our results contain many false discoveries by chance. Strategies to control false discovery rate caused by the multiple hypothesis testing problem usually adjust the significance level α in some way [8, 46]. E.g., a simple way to adjust α is called the *Bonferroni correction* [11], which uses $\frac{\alpha}{m}$ instead of α, where m is the number of tests. However, this correction can fail with the presentation of more tests with non-significant results.

Instead of adjusting the significance level α for every test, another strategy is to rank the tests by their *p-values* in an ascending order, and then select a certain number of top ranked tests as significant. This is called the FDR approach [22], which ensures that in expectation the false discovery rate is below a user-defined threshold α. Common FDR approaches are Benjamini-Hochberg (BH) and Benjamini-Yekutieli (BY) procedures [22], which differ in how many top ranked tests to declare as significant. We employ the BY procedure since it controls the FDR under arbitrary dependence assumptions, which is appropriate for this study because two tests should not be considered independent if their experiment specifications have common attributes. For each relation in $\{R1, R2, R3\}$, we conduct a separate BY procedure and use $\alpha = 0.05$.

V. Analyzing CleanML Database

We first present our strategy for performing result analysis in Section V-A, and discuss the detailed findings for each error type in Sections V-B to V-F.

A. Strategy for Result Analysis

We investigate the impact of data cleaning on ML by running SQL queries on $R \in \{R1, R2, R3\}$ (c.f. Table 1). We first present the SQL query templates, where $E \in \{$inconsistencies, duplicates, mislabels, outliers, missing values$\}$, and then discuss two angles for analyzing the results of SQL queries.

Varying Granularity of Analysis. The impact of cleaning on ML depends on a variety of factors, warranting varying granularities of analysis. We first fix the error type and group by the flag attribute (Q1), which aggregates over all datasets, ML models, scenarios, and cleaning methods. This gives the impact of cleaning the given type of error in general. Then we group by an additional attribute to see if there is any scenario (Q2), ML model (Q3), cleaning method (Q4.1 for detection and Q4.2 for repair), or dataset (Q5), where the impact is different from the general trend we observe from Q1.

Compare $R1$, $R2$, and $R3$. We also investigate the difference of results between the same query template issued against different relations. This indicates whether performing model selection (i.e., $R2$) and cleaning algorithm selection (i.e., $R3$) are helpful for achieving a positive impact when applying data cleaning on downstream ML models. Due to space limitation, throughout the remaining analysis, we will only show the results of those SQL queries issued against those relations that convey nontrivial and unique findings. We refer readers to the technical report [34] for all query results.

Q1: Flag.

```
SELECT flag, COUNT(*)
FROM R
WHERE error_type = E
GROUP BY flag
```

Q2: Scenario.

```
SELECT scenario, flag, COUNT(*)
FROM R
WHERE error_type = E
GROUP BY scenario, flag
```

Q3: ML Model (Not applicable to R2, R3).

```
SELECT ml_model, flag, COUNT(*)
FROM R
WHERE error_type = E
GROUP BY ml_model, flag
```

Q4: Clean Method (Not applicable to R3 or $E \in \{$ inconsistencies, mislabels$\}$, where only one cleaning method is applied).

Q4.1:

```
SELECT detection, flag, COUNT(*)
FROM R
WHERE error_type = E
GROUP BY detection, flag
```

Q4.2:

```
SELECT repair, flag, COUNT(*)
FROM R
WHERE error_type = E
GROUP BY repair, flag
```

Q5: Dataset.

```
SELECT dataset, flag, COUNT(*)
FROM R
WHERE error_type = E
GROUP BY dataset, flag
```

B. Missing Values

TABLE 11. Query Results for Missing Values

Q1 (E = Missing Values)

R	P	S	N
R1	49% (143)	27% (80)	24% (71)
R2	57% (24)	21% (9)	21% (9)
R3	33% (2)	50% (3)	17% (1)

Q4.2 (E = Missing Values)

R	Imputation	P	S	N
R1	HoloClean	38% (16)	29% (12)	33% (14)
	Mean_Dummy	52% (22)	24% (10)	24% (10)
	Mean_Mode	50% (21)	29% (12)	21% (9)
	Median_Dummy	52% (22)	24% (10)	24% (10)
	Median_Mode	64% (27)	31% (13)	5% (2)
	Mode_Dummy	52% (22)	24% (10)	24% (10)
	Mode_Mode	31% (13)	31% (13)	38% (16)
R2	HoloClean	50% (3)	17% (1)	33% (2)
	Mean_Dummy	67% (4)	17% (1)	17% (1)
	Mean_Mode	67% (4)	17% (1)	17% (1)
	Median_Dummy	50% (3)	33% (2)	17% (1)
	Median_Mode	83% (5)	17% (1)	0% (0)
	Mode_Dummy	50% (3)	33% (2)	17% (1)
	Mode_Mode	33% (2)	17% (1)	50% (3)

Q5 (E = Missing Values)

R	Dataset	P	S	N
R1	Airbnb	6% (3)	88% (43)	6% (3)
	BabyProduct	57% (28)	0% (0)	43% (21)
	Credit	29% (14)	67% (33)	4% (2)
	Marketing	49% (24)	8% (4)	43% (21)
	Titanic	65% (32)	0% (0)	35% (17)
	USCensus	86% (42)	0% (0)	14% (7)

Table 11 shows the results of SQL queries for missing values issued against all three relations. **(Q1):** As we can see in the results of Q1, cleaning missing values by imputation mostly improves the performance or achieves similar performance as deleting records with missing values. However, there are still quite a few "N" flags in the results of $R1$ and $R2$, which indicates that imputation may be worse than deletion. This is because values filled in by imputation algorithms are various "guesses", which may be far off the ground-truth values. Comparing the results of Q1 on $R1$, $R2$, we observe that imputation is more likely to have positive impact and less likely to have negative impact on better ML models. Comparing the results of Q1 on $R2$ and $R3$, we can see that selecting the best imputation method using a validation set can further reduce the negative impacts. **(Q2):** As mentioned in Section III-E, we only consider the scenario BD for missing values, hence Q2 is not applicable. **(Q3):** All models share similar results as we observe in Q1; a result table hence is omitted here. **(Q4):** Since there is only one detection method for missing values, Q4.1 is not applicable. From the results of Q4.2 on R1 and R2 (Q4.2 is not applicable to R3), we can see that there is no significant difference between impacts of different imputation methods. In particular, the state-of-the-art probabilistic imputation method HoloClean [44] has no clear

advantage over other simple imputation methods. **(Q5):** We show the results of Q5 issued against R1 only, as the results of Q5 issued against R2 and R3 reveal similar findings. As we can see, the impact of data cleaning varies significantly across different datasets. For example, the "USCensus" dataset has more "P"%, while "Airbnb" dataset has more "S"%.

Summary of Observations for Missing Values. (1) Cleaning missing values by imputation is more likely to improve ML or achieve similar performance compared with deletion; (2) with model and imputation method selection, we are more likely to observe positive impacts and less likely to observe negative impacts; (3) impacts vary largely across different datasets and advanced cleaning methods such as HoloClean [44] is not noticeably better than simple imputation methods.

C. Outliers

TABLE 12. Query Results for Outliers

Q1 (E = Outliers)

R	P	S	N
R1	31% (176)	61% (339)	8% (45)
R2	39% (31)	56% (45)	5% (4)
R3	12% (1)	88% (7)	0% (0)

Q3 (E = Outliers)

R	Model	P	S	N
R1	Adaboost	12% (10)	70% (56)	18% (14)
	Decision Tree	30% (24)	69% (55)	1% (1)
	Gussian Naive Bayes	31% (25)	64% (51)	5% (4)
	KNN	52% (42)	42% (34)	5% (4)
	Logistic Regression	22% (18)	60% (48)	18% (14)
	Random Forest	32% (26)	60% (48)	8% (6)
	XGboost	39% (31)	59% (47)	2% (2)

Q4.1 (E = Outliers)

R	Detect	P	S	N
R1	IF	34% (57)	47% (79)	19% (32)
	IQR	59% (99)	38% (64)	3% (5)
	SD	8% (13)	90% (151)	2% (4)
R2	IF	38% (9)	58% (14)	4% (1)
	IQR	71% (17)	17% (4)	12% (3)
	SD	17% (4)	83% (20)	0% (0)

Q4.2 (E = Outliers)

R	Repair	P	S	N
R1	HoloClean	12% (7)	80% (45)	7% (4)
	Mean	33% (56)	60% (101)	7% (11)
	Median	33% (56)	58% (97)	9% (15)
	Mode	34% (57)	57% (96)	9% (15)
R2	HoloClean	12% (1)	88% (7)	0% (0)
	Mean	42% (10)	54% (13)	4% (1)
	Median	46% (11)	50% (12)	4% (1)
	Mode	38% (9)	54% (13)	8% (2)

Q5 (E = Outliers)

R	Dataset	P	S	N
R1	Airbnb	10% (14)	87% (122)	3% (4)
	Credit	14% (20)	70% (98)	16% (22)
	EEG	57% (80)	41% (57)	2% (3)
	Sensor	44% (62)	44% (62)	11% (16)

Table 12 shows the query results for outliers issued against all three relations. **(Q1):** We can see that cleaning outliers mostly has an insignificant impact on model performance. This is in contrast to the findings for missing values, where cleaning mostly have a positive impact. We believe that it is because cleaning outliers is a fundamentally harder task — while detecting missing values is trivial, detecting outliers is prone to mistakes, which may declare non-outliers as outliers and miss true outliers. Comparing the results of R1, R2 and R3 shows that with model selection and cleaning method selection, "N"% gradually decreases to 0. This indicates that using model and cleaning method selection can eliminate some negative impacts and improve robustness without losing much benefit, which is similar to the findings for missing values. **(Q2):** The findings of Q2 are similar to Q1, and hence are omitted. **(Q3):** KNN has mostly "P" flags, less "N" and "S" flags than most of the other ML models. It is because that KNN relies on the distances between examples to make predictions, and outliers can impact distances significantly. The differences between other models are not prominent. **(Q4.1):** As we see in Q4.1 on R1, IF and IQR have more "P" flags and "N" flags than the SD method. This indicates that IF and IQR are more aggressive than SD. Indeed, by examining the number of outliers detected by these methods, we learn that IF and IQR generally produce many more outliers than SD. **(Q4.2):** The results of Q4.2 show that there is no significant difference between repair methods in R1 and R2. Similar to missing values, it suggests that advanced data cleaning methods such as HoloClean [44] are not noticeable better than simple cleaning methods in terms of improving downstream ML models. **(Q5):** We only show results of Q5 on R1, as the results on R2 and R3 show similar findings. We can see that most negative flags are from the "Credit" dataset. In all relations, "EEG" and "Sensor" have more "P" flags than other datasets. This echoes our observations for missing values: the impact of cleaning varies significantly across datasets. However, for outliers, we are not able to provide more detailed explanations as we do not even know whether the detected outliers are true outliers.

Summary of Observations for Outliers. (1) Cleaning outliers is more likely to have insignificant impact on model performance, at least for the set of outlier cleaning algorithms we considered; (2) with model selection and cleaning algorithm selection, the probability of having negative impacts can be greatly reduced; (3) the impact of cleaning varies vastly across datasets; (4) since outliers are harder to clean, different detection methods have major differences in observed impact.

D. Mislabels

Table 13 shows the query results for mislabels. **(Q1):** We observe that cleaning mislabels mostly improves or insignificantly affects the performance of ML models. However, there are still a few "N" flags in R1 and R2. This is because the automatic mislabel cleaning method cleanlab can make mistakes in detecting and correcting mislabels, which may harm the model performance. Comparing the results of R1 and R2, with model selection (R2), it becomes more likely to have positive impacts. **(Q2):** We observe that in scenario BD, it is more likely to have insignificant impact, while it is more likely to have positive impact in scenario CD. This is because mislabels in the test set can directly change the accuracy (flipping labels can directly make true positives become false positives). However, mislabels in the training set affect the final test accuracy indirectly. Hence, the impact is less significant in BD than that in CD. **(Q3):** We notice that AdaBoost and XGBoost are more reactive to mislabels, because boosting procedures assign higher weights to on those instances that are predicted incorrectly. Therefore, cleaning mislabeled examples largely alleviates negative impacts on boosting ML. **(Q4):** Q4 is not applicable for mislabels because

TABLE 13. Query Results for Mislabel

Q1 (E = Mislabel)				
R		P	S	N
R1		47% (85)	38% (70)	15% (27)
R2 & R3		54% (14)	31% (8)	15% (4)
Q2 (E = Mislabel)				
R	Scenario	P	S	N
R1	BD	19% (17)	59% (54)	22% (20)
	CD	75% (68)	18% (16)	8% (7)
R2 & R3	BD	23% (3)	46% (6)	31% (4)
	CD	85% (11)	15% (2)	0% (0)
Q3 (E = Mislabel)				
R	Model	P	S	N
R1	Adaboost	54% (14)	31% (8)	15% (4)
	Decision Tree	46% (12)	46% (12)	8% (2)
	Gussian Naive Bayes	38% (10)	42% (11)	19% (5)
	KNN	38% (10)	42% (11)	19% (5)
	Logistic Regression	50% (13)	38% (10)	12% (3)
	Random Forest	50% (13)	31% (8)	19% (5)
	XGboost	50% (13)	38% (10)	12% (3)
Q5 (E = Mislabel)				
R	Dataset	P	S	N
R1	Clothing	21% (3)	14% (2)	64% (9)
	EEG_major	43% (6)	29% (4)	29% (4)
	EEG_minor	50% (7)	29% (4)	21% (3)
	EEG_uniform	71% (10)	29% (4)	0% (0)
	Marketing_major	57% (8)	43% (6)	0% (0)
	Marketing_minor	86% (12)	14% (2)	0% (0)
	Marketing_uniform	64% (9)	36% (5)	0% (0)
	Titanic_major	21% (3)	79% (11)	0% (0)
	Titanic_minor	7% (1)	79% (11)	14% (2)
	Titanic_uniform	50% (7)	21% (3)	29% (4)
	USCensus_major	43% (6)	36% (5)	21% (3)
	USCensus_minor	43% (6)	50% (7)	7% (1)
	USCensus_uniform	50% (7)	43% (6)	7% (1)

we have only one way of cleaning, i.e., cleanlab. **(Q5):** We observe that the impact varies vastly across datasets. For example, in "Marketing_major" dataset, there are more "P"%, while in "Clothing" dataset, there are more "N"%. The results of Q5 issued against R2 and R3 (omitted here) reveal similar findings to that of Q1.

Summary of Observations for Mislabels: (1) Cleaning mislabels is likely to have positive or insignificant impacts on ML; (2) With model selection, it is more likely to have positive impacts. (3) boosting based ML models are most reactive to mislabels. (4) the impact varies significantly across datasets.

E. Inconsistencies

TABLE 14. Query Results for Inconsistencies

Q1 (E = Inconsistencies)				
R		P	S	N
R1		12% (7)	88% (49)	0% (0)
R2 & R3		25% (2)	75% (6)	0% (0)
Q5 (E = Inconsistencies)				
R	Dataset	P	S	N
R1	Company	29% (4)	71% (10)	0% (0)
	Movie	14% (2)	86% (12)	0% (0)
	Restaurant	0% (0)	100% (14)	0% (0)
	University	7% (1)	93% (13)	0% (0)

Table 14 shows the query results for inconsistencies. **(Q1):** The results of Q1 shows no negative impact when cleaning inconsistencies, though most experiments show insignificant impact. Furthermore, comparing results of $R1$ with $R2$, selecting the best ML model increases the likelihood of observing positive impact after cleaning inconsistencies. **(Q2, Q3, Q4):** Grouping by the additional scenario attribute Q2 and ML model attribute Q3 reveals similar findings as Q1. Since we only have one cleaning method for inconsistencies, Q4 is not applicable. Their results are hence omitted. **(Q5):** From the results of Q5, in general, the pattern holds that insignificant impact of cleaning inconsistency prevails and no negative impacts of cleaning inconsistency is found. However, distribution of "P" flags and "N" flags varies significantly across datasets. For example, "Company" and "Movie" datasets have more "P" flags due to the much greater number of inconsistencies in these two datasets upon examining the cleaning results.

Summary of Observations for Inconsistencies: (1) Cleaning inconsistencies is more likely to have insignificant impact and unlikely to have negative impact on ML; (2) the impact varies significantly across datasets.

F. Duplicates

TABLE 15. Query Results for Duplicates

Q1 (E = Duplicates)				
R		P	S	N
R1		11% (12)	67% (75)	22% (25)
R2		12% (2)	56% (9)	31% (5)
R3		12% (1)	50% (4)	38% (3)
Q4.1 (E = Duplicates)				
R	Detection	P	S	N
R1	ZeroER	5% (3)	61% (34)	34% (19)
	Key Collision	16% (9)	73% (41)	11% (6)
R2	ZeroER	12% (1)	50% (4)	38% (3)
	Key Collision	12% (1)	62% (5)	25% (2)
Q5 (E = Duplicates)				
R	Dataset	P	S	N
R1	Airbnb	4% (1)	86% (24)	11% (3)
	Citation	11% (3)	71% (20)	18% (5)
	Movie	29% (8)	21% (6)	50% (14)
	Restaurant	0% (0)	89% (25)	11% (3)

Table 15 shows the query results for duplicates. **(Q1):** In all the relations, there are more "S" flags and "N" flags than "P" flags. Upon examining the detected duplicates, we find that duplicate detection algorithms may produce many false positives, where some non-duplicated examples are incorrectly identified as duplicates. When these non-duplicated examples are removed from the training set, useful information may be lost, which can negatively affect the model performance. **(Q2, Q3):** Grouping by additional scenarios Q2 or additional ML models Q3 reveals similar findings to Q1. Hence, findings on Q2 and Q3 are omitted here. **(Q4.1):** From results of Q4.1, we observed that in both R1 and R2, detection using ZeroER is relatively more likely to have negative impacts compared to detection with key collision method. This is because ZeroER is more aggressive and also produces more false positives than key collision detection on the datasets upon examining the cleaning results. **(Q4.2):** Q4.2 is not applicable to duplicates because there is only one repair method (deletion). **(Q5):** We only show the results of Q5 issued against R1, as the results on R2 and R3 reveal similar findings. We observed that the impact varies vastly across dataset. This is consistent with other error types, and is mainly due to the different error distributions each dataset may exhibit.

Summary of Observations for Duplicates: (1) Cleaning duplicates is more likely to have insignificant or negative impacts than positive impacts; (2) the impact on cleaning duplicates varies vastly across detection methods and datasets.

VI. Overall Observations for Single Error Types

In Table 16, we summarize the overall findings from Sections V and discuss them in this section.

TABLE 16. Summary of Empirical Findings for Single Error Types

Error Type	Impact on ML	Does the impact depend on			
		Datasets	Scenarios	Cleaning Algos	ML Algorithms
Duplicates	Varying (Mostly S & N)		No	Yes	No
Inconsistencies	Varying (Mostly S)		No	N.A.	No
Missing Values	Varying (Mostly P & S)	Yes	No	Yes	No
Mislabels	Varying (Mostly P & S)		Yes	N.A.	No (except Boosting)
Outliers	Varying (Mostly S)		No	Yes	No (except KNN)

Strong Dependency on Dataset. While the impact of cleaning on ML is hardly consistent, we always observe that there are noticeable differences between results of Q1 and Q5 in terms of the distributions of "P", "N" and "S" impacts for all error types. This suggests that the cleaning impact depends on datasets — while two datasets may contain errors of the same type, the distributions of those errors can be vastly different. Therefore, practitioners should never make arbitrary cleaning decisions dealing with dirty data in ML classification tasks.

Consistent Impact from Cleaning w.r.t. Model and Scenario. While some ML models are noticeably more sensitive to some error types (e.g., KNN to outliers in Table 12 Q3 and Adaboost to mislabels in Table 13 Q3) , we observe that usually there is no notable difference between results of Q1 and that of Q3 in terms of the distribution of "P", "N", and "S" impact, for all error types. In other words, if cleaning a dataset has a particular impact for one ML model, cleaning is likely to have the same type of impact for other models as well. This suggests that in the presence of dirty data in ML classification tasks, performing data cleaning is a more broadly applicable solution, compared with developing specific robust ML models. Similarly, the difference between the results of Q1 and that of Q2 in terms of the distribution of "P", "N", and "S" is also largely negligible (except for mislabels in Section V-D). This suggests that cleaning can be valuable in both the model development and model deployment phase.

Strong Dependency on Cleaning Algorithms. Because different datasets can have different error distributions (even for the same error type), no single automatic cleaning algorithm is always the best. Regardless, on all error types, we observe the same patterns comparing the results of all five queries against $R1$, $R2$, and $R3$: (1) data cleaning is more likely to have a positive impact on better ML models, i.e., models chosen by a validation set; and (2) the cleaning algorithm chosen by a validation set is more likely to have a positive impact than a randomly chosen cleaning algorithm. In practice, selecting a data cleaning algorithm using a validation set is sensible. However, one should not treat this as a golden solution as we do observe cases, where the selected cleaning algorithm can still degrade ML model performances (e.g., Table 15 Q1).

VII. Mixed Errors, RobustML and Human Cleaning

The previous section analyzes the impact of automatic cleaning methods for different single error types on various downstream ML models. In this section, we perform additional experiments to study (1) the effect of cleaning multiple error types simultaneously; (2) how does the approach of cleaning for ML compare with robust ML approaches; and (3) does human cleaning benefit downstream models more compared with automatic cleaning methods?

A. Cleaning Mixed Error Types

Setup: For datasets with multiple error types, we study if there is benefit in cleaning them all compared with cleaning a single error type. The cleaning algorithm space considered for cleaning multiple error types is the Cartesian product of cleaning algorithms for each component error type as listed in Table 2. For each dataset with multiple real error types (c.f. Table 3), we compare the best model obtained by cleaning all error types with that obtained by cleaning a single error type, i.e., the best model obtained after model selection and cleaning algorithm selection (c.f. R3 in the CleanML schema). For each dataset, we use 20 train/test splits, and perform statistical hypothesis testing, as described in Section IV-B, to obtain a flag, P, S, or N — indicating cleaning multiple error types is better than, has no significant difference from, or is worse than cleaning a single error type, respectively.

Results: Table 17 shows the results[3], where we can observe the following. (1) Cleaning all error types in a dataset is not always better than cleaning a single error type. This is similar to the impact we observe for cleaning a single error type, which does not always result in a better model compared with no cleaning and is dataset dependent. (2) The chance of having a negative impact when cleaning multiple error types is also very low. The only negative case we found is cleaning inconsistency + duplicates can be worse than cleaning inconsistency only. This is not surprising, since we discover in Section V-F that cleaning duplicates is likely to bring negative impacts. (3) On top of any single error type, additionally cleaning missing values or outliers is likely to bring positive impacts and has no negative impacts in all cases.

TABLE 17. Cleaning Mixed Error Types vs. Single Error Type

Dataset	Mixed Error Types	Single Error Type	P	S	N
Credit	Missing Values + Outliers	Outliers	100% (1)	0% (0)	0% (0)
		Missing Values	100% (1)	0% (0)	0% (0)
Restaurant, Movie	Inconsistency + Duplicates	Inconsistency	0% (0)	0% (0)	100% (2)
		Duplicates	50% (1)	50% (1)	0% (0)
Airbnb	Missing Values + Outliers + Duplicates	Outliers	0% (0)	100% (1)	0% (0)
		Missing Values	0% (0)	100% (1)	0% (0)
		Duplicates	100% (1)	0% (0)	0% (0)

B. CleanML v.s. Robust ML Approaches

Robust ML. Instead of performing data cleaning to deal with noisy data, the ML community mostly focuses on developing ML algorithms robust to some particular noise type with certain distributions (e.g., [41] on noise-robust decision trees, [3] on decision trees against label noise, [48] on regularization to improve the robustness of ML models, [31] on missing values, [38] on mislabels). This highlights a clear advantage of data cleaning, which can be used to handle any error types and cleaned datasets can be used to train any end models without modifications to training procedures.

Setup: As a case study, we compare CleanML with the open-source NaCL [31] package that develops a specialized form of Logistic Regression model that is robust to missing values. In addition, for all other error types, we compare CleanML with deep learning (DL) models as a form of robust ML, as they are

[3]Note that we do not consider mixed error types incl. mislabels, as we do not have datasets with coexisting real mislabels and other errors.

generally less sensitive to various data errors than conventional ML models (e.g., DL robustness to massive label noise [45] and to input corruptions [27]). The DL model is a Multi-layer Perceptron classifier (MLP) with three layers and we train the model using *optuna*[4]. For each dataset, we compare the best model obtained through data cleaning, i.e., by model and cleaning algorithm selection, with a robust end model. For each dataset, we used 20 train/test splits, and performed statistical hypothesis testing, as described in Section IV-B, to obtain a flag, P, S, or N — indicating that data cleaning is better than, has no significant difference from, or is worse than using robust ML approaches, respectively.

Results: We discuss the results in Table 18. (1) For many cases, data cleaning leads to a better end model compared with robust ML. (2) For missing values, we compare NaCL (a robust LR model) with a regular LR model under the best data cleaning algorithm, we find that cleaning is overall better (row 1). Comparing NaCL with the best model under the best cleaning algorithm, the benefit of data cleaning over robust ML widens (row 2). This exactly highlights the merit of data cleaning, i.e., data cleaning can be performed for any error type and cleaned datasets can be used to train any end model, while robust ML algorithms are usually specifically designed for an error type and a specific model type. (3) Duplicates is the only error type where deep learning is overall better than cleaning, which can be explained by our previous finding that cleaning duplicates is likely to harm ML models.

TABLE 18. Robust ML vs. Data Cleaning

Data Cleaning for ML	RobustML	Error Type	P	S	N
LR + Best Cleaning Alg	NACL	Missing Values	33% (2)	50% (3)	17% (1)
Best Model + Best Cleaning Alg	NACL	Missing Values	83% (5)	0% (0)	17% (1)
Best Model + Best Cleaning Alg	MLP	Mislabel	85% (11)	15% (2)	0% (0)
		Inconsistency	50% (2)	50% (2)	0% (0)
		Outliers	50% (2)	25% (1)	25% (1)
		Duplicates	0% (0)	75% (3)	25% (1)

C. Human Cleaning vs. Automatic Cleaning Algorithms

Setup: All cleaning algorithms considered in Table 2 are automatic cleaning algorithms in that they require no or minimal human involvement in tuning/setting up the cleaning algorithms. Here, we study if human cleaning benefits ML models compared with automatic cleaning, e.g., by spending efforts in obtaining ground-truth values for missing values or mislabels or by spending efforts in designing data quality rules for fixing inconsistencies. For each dataset, we compare the best model under human cleaning with the best model using the best automatic cleaning method. Again, we use 20 train/test splits and leverage statistical testing as described in Section IV-B, to obtain a flag, P, S, or N — indicating that human cleaning is better than, has no significant difference from, or is worse than automatic cleaning, respectively.

Results: (1) As we can see in Table 19, for the two datasets "BabyProduct" and "Clothing", where humans manually filled missing values and corrected mislabels, the results of human cleaning are better than the best automatic cleaning method. (2) For the three datasets with inconsistencies, we manually curate data quality rules in the form of denial constraints, which are then used to detect and repair inconsistencies. We observe that the results from rule-based cleaning have no significant difference from using automatic cleaning methods.

TABLE 19. Automatic Cleaning vs. Human Cleaning

Dataset	Error Type	P	S	N
BabyProduct	Missing Values	100% (1)	0% (0)	0% (0)
Clothing	Mislabel	100% (1)	0% (0)	0% (0)
Company, Restaurant, University	Inconsistencies	0% (0)	100% (3)	0% (0)

VIII. Future Research

Our study suggests that data cleaning can greatly improve downstream ML performance in many cases. This suggests that there are many opportunities for future research in the area of cleaning for ML. We share some of our thoughts.

Improving the Study. Our current study on data cleaning for ML can be further improved/augmented in multiple ways. (1) While we focus on classification tasks, future studies could study how various errors affect other ML tasks, such as regression tasks and unsupervised clustering. (2) While we include many datasets with real-world errors for studying the impacts of automatic cleaning, more real-world datasets with ground truth could strength the study on human cleaning (e.g., one can use two versions of the same dataset, where the newer version can serve as the truth for the older one).

Theoretical Formalization. While we provide many interesting empirical explanations for the observed impacts of cleaning on ML, it is extremely valuable to design a theoretical framework that can precisely quantify the impacts. The DB community has long proposed the notion of *consistent query answering* [6, 9], which aims to answer SQL queries in the presence of dirty data. In consistent query answering, a tuple is included in the result of a query against the dirty database D only if that tuple appears in the result of a query issued against every possible cleaned version of D. Under this framework, different classes of SQL queries and different cleaning semantics have been studied, and they entail different computation complexities for query answering. We believe that a similar theoretical framework that quantifies the impact of cleaning on ML is a fundamental step. For example, one could similarly define a notion of *consistent machine learning*, where cleaning can be deemed unnecessary if the prediction of a tuple is identical for every possible cleaned version of D. Rigorously defining a theoretical framework for ML in the presence of dirty data is more challenging than that for SQL — while answering an SQL query is deterministic, the ML model training process is inherently probabilistic. Nonetheless, once a first attempt is taken, we expect to see many followup works that focus on different cleaning semantics and ML models.

Better Cleaning Solutions. While our current study uses a fixed set of cleaning algorithms, there are ample research opportunities to design new cleaning approaches specifically for ML. First, we could design better automatic cleaning algorithms. The lack of ground-truth (i.e., labeled examples) has been a long-standing challenge in designing general-purpose data cleaning solutions, which is why existing data cleaning algorithms use various proxy objectives (e.g., minimality of

[4]We tune hidden layer size, learning rate, momentum, and optimizer for each data split. The activation function is *ReLu* and each model runs for 100 epochs. About *optuna* refer to https://github.com/optuna/optuna.

repairs [10, 15]). Fortunately, in the problem of data cleaning for ML, we have a more clearly defined objective, i.e., to improve the downstream ML model performance. Therefore, designing new cleaning algorithms that maximize the ML model performance is a promising direction. We anticipate the primary challenge of this direction is to avoid training the ML models many times to select the best candidate. When multiple error types co-occur, designing automatic cleaning becomes more challenging as the space of cleaning methods becomes larger, and there may exist non-trivial interaction/dependency between different error types or cleaning methods that our current study has not considered. Second, we could design cleaning solutions that involve human cleaners. Our study indicates that human cleaning, especially when directly correcting data errors, can lead to even better ML models compared to automatic cleaning. However, we must also minimize/prioritize human cleaning efforts, (e.g., ActiveClean [33] via active learning, CPClean [29] based on certain predictions), where humans are asked to clean the most beneficial examples first.

IX. Acknowledgement

The Chu Data Lab acknowledges the support from SCS in GT, Georgia State funds, as well as the JP Morgan Faculty award. CZ and the DS3Lab gratefully acknowledge the support from the Swiss National Science Foundation (Project Number 200021_184628), Innosuisse/SNF BRIDGE Discovery (Project Number 40B2-0_187132), European Union Horizon 2020 Research and Innovation Programme (DAPHNE, 957407), Botnar Research Centre for Child Health, Swiss Data Science Center, Alibaba, Cisco, eBay, Google Focused Research Awards, Oracle Labs, Swisscom, Zurich Insurance, Chinese Scholarship Council, and the Department of Computer Science at ETH Zurich.

References

[1] Cleaning big data: Most time-consuming, least enjoyable data science task. https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/.

[2] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *PVLDB*, 9(12):993–1004, 2016.

[3] J. Abellán and S. Moral. Building classification trees using the total uncertainty criterion. *IJIS*, 18(12):1215–1225, 2003.

[4] C. C. Aggarwal. *Outlier Analysis*. Springer, 2013.

[5] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. QSGD: Communication-efficient sgd via gradient quantization and encoding. In *NIPS*, pages 1709–1720, 2017.

[6] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. pages 68–79, 1999.

[7] P. C. Arocena, B. Glavic, G. Mecca, R. J. Miller, P. Papotti, and D. Santoro. Messing up with bart: error generation for evaluating data-cleaning algorithms. *PVLDB*, 9(2):36–47, 2015.

[8] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B*, 57(1):289–300, 1995.

[9] L. E. Bertossi. *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers, 2011.

[10] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. pages 746–755, 2007.

[11] C. E. Bonferroni. Teoria statistica delle classi e calcolo delle probabilità. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze*, 8:3–62, 1936.

[12] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *22nd KDD*, pages 785–794. ACM, 2016.

[13] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang. Data cleaning: Overview and emerging challenges. In *2016ICMD*, pages 2201–2206, 2016.

[14] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.

[15] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *2013ICDE*, pages 458–469. IEEE, 2013.

[16] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *2015KDD*, pages 1247–1261. ACM, 2015.

[17] C. De Sa, M. Feldman, C. Ré, and K. Olukotun. Understanding and optimizing asynchronous low-precision stochastic gradient descent. In *ACM SIGARCH Computer Architecture News*, volume 45, pages 561–574. ACM, 2017.

[18] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang. Distributed representations of tuples for entity resolution. *PVLDB*, 11(11):1454–1467, 2018.

[19] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1):1–16, 2007.

[20] W. Fan and F. Geerts. Foundations of data quality management. *Synthesis Lectures on Data Management*, 4(5):1–217, 2012.

[21] B. Frénay and M. Verleysen. Classification in the presence of label noise: a survey. *TNNLS*, 25(5):845–869, 2014.

[22] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[23] S. García, J. Luengo, and F. Herrera. *Data preprocessing in data mining*. Springer, 2015.

[24] Y. He, X. Chu, K. Ganjam, Y. Zheng, V. Narasayya, and S. Chaudhuri. Transform-data-by-example (tde): an extensible search engine for data transformations. *PVLDB*, 11(10):1165–1177, 2018.

[25] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas. Holodetect: Few-shot learning for error detection. In P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, editors, *2019ICMD*, pages 829–846. ACM, 2019.

[26] J. M. Hellerstein. Quantitative data cleaning for large databases. *UNECE*, 2008.

[27] D. Hendrycks and T. G. Dietterich. Benchmarking neural network robustness to common corruptions and surface variations. *arXiv*, 2018.

[28] I. F. Ilyas, X. Chu, et al. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends® in Databases*, 5(4):281–393, 2015.

[29] B. Karlaš, P. Li, R. Wu, N. M. Gürel, X. Chu, W. Wu, and C. Zhang. Nearest neighbor classifiers over incomplete information: From certain answers to certain predictions. *arXiv preprint arXiv:2005.05117*, 2020.

[30] T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano. Comparing boosting and bagging techniques with noisy and imbalanced data. *IEEE SMC*, 41(3):552–568, 2011.

[31] P. Khosravi, Y. Liang, Y. Choi, and G. V. d. Broeck. What to expect of classifiers? reasoning about logistic regression with missing features. *arXiv*, 2019.

[32] S. Krishnan, M. J. Franklin, K. Goldberg, and E. Wu. Boostclean: Automated error detection and repair for machine learning. *arXiv*, 2017.

[33] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: interactive data cleaning for statistical modeling. *PVLDB*, 9(12):948–959, 2016.

[34] P. Li, X. Rao, J. Blase, Y. Zhang, X. Chu, and C. Zhang. CleanML Technical Report. https://github.com/chu-data-lab/CleanML/blob/master/TechReport.pdf, 2019.

[35] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *NIPS*, pages 5330–5340, 2017.

[36] J. H. McDonald. *Handbook of biological statistics*, volume 2. sparky house publishing Baltimore, MD, 2009.

[37] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra. Deep learning for entity matching: A design space exploration. In *2018ICMD*, pages 19–34. ACM, 2018.

[38] C. G. Northcutt, T. Wu, and I. L. Chuang. Learning with confident examples: Rank pruning for robust classification with noisy labels. In *33rd UAI*, UAI'17. AUAI Press, 2017.

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830, 2011.

[40] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich. Data management challenges in production machine learning. In *2017ICMD*, pages 1723–1726, 2017.

[41] J. R. Quinlan. Simplifying decision trees. *IJMMS*, 27(3):221–234, 1987.

[42] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[43] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, pages 693–701, 2011.

[44] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB*, 10(11):1190–1201, 2017.

[45] D. Rolnick, A. Veit, S. Belongie, and N. Shavit. Deep learning is robust to massive label noise. *arXiv*, 2017.

[46] G. Rupert Jr et al. *Simultaneous statistical inference*. Springer Science & Business Media, 2012.

[47] S. K. Sarkar, H. Midi, and S. Rana. Detection of outliers and influential observations in binary logistic regression: An empirical study. *Journal of Applied Sciences*, 11(1):26–35, 2011.

[48] C. M. Teng. Evaluating noise correction. In *PRICAI*, pages 188–198. Springer, 2000.

[49] R. Verborgh and M. De Wilde. *Using OpenRefine*. Packt Publishing Ltd, 2013.

[50] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *2014ICMD*, pages 469–480. ACM, 2014.

[51] R. Wu, S. Chaba, S. Sawlani, X. Chu, and S. Thirumuruganathan. Zeroer: Entity resolution using zero labeled examples. In *2020 ICMD*, pages 1149–1164, 2020.

[52] H. Zhang, J. Li, K. Kara, D. Alistarh, J. Liu, and C. Zhang. Zipml: Training linear models with end-to-end low precision, and a little bit of deep learning. In *34th ICML*, pages 4035–4043. JMLR. org, 2017.

[53] W. Zhang, S. Gupta, X. Lian, and J. Liu. Staleness-aware async-sgd for distributed deep learning. *arXiv*, 2015.

[54] Z. Zhao, L. De Stefani, E. Zgraggen, C. Binnig, E. Upfal, and T. Kraska. Controlling false discoveries during interactive data exploration. In *2017ICMD*, pages 527–540. ACM, 2017.