Auto-Test: Learning Semantic-Domain Constraints for Unsupervised Error Detection in Tables

QIXU CHEN*, Hong Kong University of Science and Technology, China
YEYE HE†, Microsoft Research, USA
RAYMOND CHI-WING WONG, Hong Kong University of Science and Technology, China
WEIWEI CUI, Microsoft Research, China
SONG GE, Microsoft Research, China
HAIDONG ZHANG, Microsoft Research, China
DONGMEI ZHANG, Microsoft Research, China
SURAJIT CHAUDHURI, Microsoft Research, USA

Data cleaning is a long-standing challenge in data management. While powerful logic and statistical algorithms have been developed to detect and repair data errors in tables, existing algorithms predominantly rely on domain-experts to first manually specify data-quality constraints specific to a given table, before data cleaning algorithms can be applied.

In this work, we propose a new class of data-quality constraints that we call *Semantic-Domain Constraints*, which can be reliably inferred and automatically applied to *any tables*, without requiring domain-experts to manually specify on a per-table basis. We develop a principled framework to systematically learn such constraints from table corpora using large-scale statistical tests, which can further be distilled into a core set of constraints using our optimization framework, with provable quality guarantees. Extensive evaluations show that this new class of constraints can be used to both (1) directly detect errors on real tables in the wild, and (2) augment existing expert-driven data-cleaning techniques as a new class of complementary constraints.

Our extensively labeled benchmark dataset with 2400 real data columns, as well as our code are available at https://github.com/qixuchen/AutoTest to facilitate future research.

CCS Concepts: • **Information systems → Data cleaning**; **Enterprise information systems**.

Additional Key Words and Phrases: Data Cleaning, Semantic Domain, Domain Constraint, Semantic Type, Data Quality, Error Detection, Unsupervised Learning, Statistical Tests

ACM Reference Format:
Qixu Chen, Yeye He, Raymond Chi-Wing Wong, Weiwei Cui, Song Ge, Haidong Zhang, Dongmei Zhang, and Surajit Chaudhuri. 2025. Auto-Test: Learning Semantic-Domain Constraints for Unsupervised Error Detection in Tables. *Proc. ACM Manag. Data* 3, 3 (SIGMOD), Article 133 (June 2025), 27 pages. https://doi.org/10.1145/3725396

*Part of work done while at Microsoft Research, email: qchenax@connect.ust.hk
†Correspondence: yeyehe@microsoft.com

Authors' Contact Information: Qixu Chen, Hong Kong University of Science and Technology, China; Yeye He, Microsoft Research, USA; Raymond Chi-Wing Wong, Hong Kong University of Science and Technology, China; Weiwei Cui, Microsoft Research, China; Song Ge, Microsoft Research, China; Haidong Zhang, Microsoft Research, China; Dongmei Zhang, Microsoft Research, China; Surajit Chaudhuri, Microsoft Research, USA.

ACM 2836-6573/2025/6-ART133
https://doi.org/10.1145/3725396

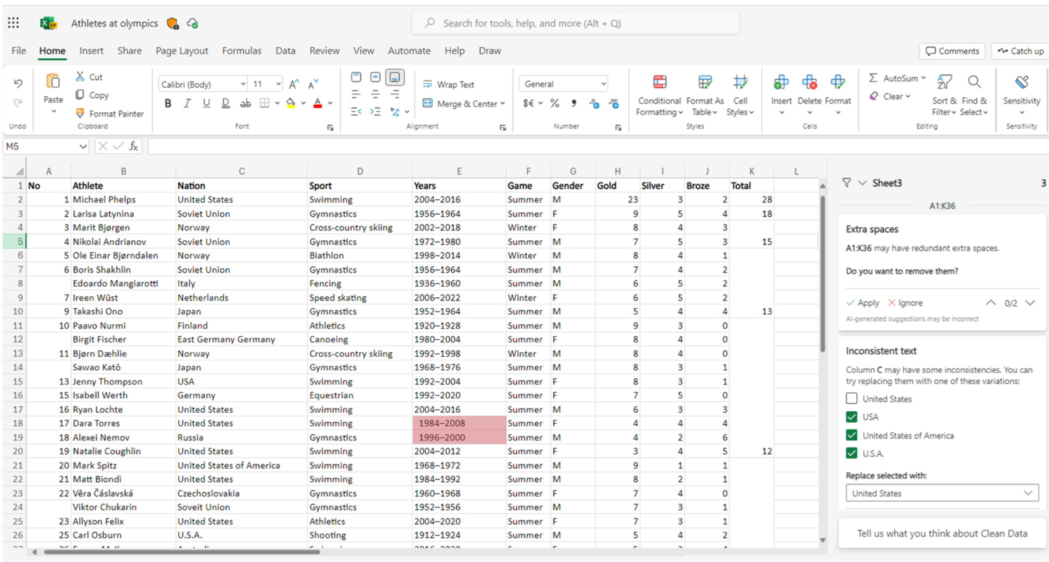


Fig. 1. Example data cleaning feature for end-users in Microsoft Excel. Data quality issues in user tables are automatically detected using techniques such as [17, 32, 69, 73], and are presented as intuitive "suggestion cards" on the side-pane (right), for users to review and accept. [This link][3] gives an end-to-end demo of how users can leverage such automated capabilities to easily clean data (without needing to define any constraints first), while staying in full control over any suggested changes that may be applied to their data.

1 Introduction

Data cleaning is a long-standing challenge in the data management community. While there is a long and fruitful line of research that developed powerful techniques using *data-quality constraints* (e.g., FD, CFD, etc.) to detect and repair data errors in tables [13, 19, 28, 39, 39, 46, 61, 74], existing methods largely depend on domain-experts to first specify data-quality constraints that are specific to a given table, before data-cleaning algorithms can be performed (while constraint discovery methods also exist, they are mainly intended to discover *candidate rules* that still require humans to verify [12, 20, 26, 33, 72]). We term this class of sophisticated and powerful approaches as "*expert-driven data cleaning*".

While such expert-driven approaches to data cleaning are extremely powerful, when experts are available to inspect each table and define relevant constraints, we observe that there is an emerging class of "*end-user data-cleaning*" use cases that aim to democratize data-cleaning for the average non-technical users, by working out-of-the-box and without requiring experts to be involved.

For example, in end-user spreadsheet tools such as Microsoft Excel [8] and Google Sheets [7] that are used by billions of non-technical users, there is a growing need to automatically detect and repair data errors in user tables out-of-the-box, *without requiring users to define constraints or provide labeled data first.*

Figure 1 shows a screenshot of such a feature in Excel, which uses techniques such as [17, 32, 69, 73] to automatically detected data-quality issues are presented as "suggestion cards" shown on the side-pane, that users can easily review and accept with the click of a button (without needing to define any constraints first). Google Sheets has a similar feature for error-detection [6].

In this work, we study a new class of data-quality constraints previously overlooked in the literature, which we call *Semantic-Domain Constraints (SDC).* Importantly, such constraints can be

	A	B	C	D	E	F	G	H	I	J	K
1	**C1 (country)**	**C2 (state code)**		**C3 (month)**	**C4 (city)**		**C5 (fiscal year)**	**C6 (unit)**		**C7 (date)**	**C8 (url)**
2	Germany	FL		january	mankato		fy17	12 oz		12/3/2020	https://twitter.com/#!/nyctbus/status/803706869944565760
3	Austria	AZ		febuary	st peter		fy18	9.8 oz		11/5/2020	https://twitter.com/#!/jrwolfson/status/799056736661475328
4	France	CA		march	seattle		fy19	0.05%		2/5/2021	https://twitter.com/#!/bmrbreakingnews/status/799061681750110208
5	Liechstein	OK		april	saint paul		fy20	28 oz		10/23/2020	__/status/799512626703323140
6	Italy	Germany		may	shakopee		fy definition	1.5 oz		10/7/2020	https://twitter.com/#!/polmsted/status/799087394884636672
7	Switzerland	AL		june	phoenix		fy21	30 oz		new facility	https://twitter.com/#!/yoadamboy/status/799108555869655046
8	Porland	GA		july	farimont		fy22	18 oz		3/26/2021	https://twitter.com/#!/yoadamboy/status/799108523535859712
9	...	...		...	...		...	...		...	...

Fig. 2. Real examples of table columns, each representing a distinct "semantic domain" (annotated in the column header). Each column C_i has a real data error (which may be a typo, or a semantically incompatible value), that is detected by a corresponding "semantic domain constraint" r_i in Table 1, which are constraints automatically learned from running AUTO-TEST.

Table 1. Example Semantic-Domain Constraints (SDCs), instantiated using CTA-classifiers (in r_1, r_2), text-embedding (r_3, r_4), regex-patterns (r_5, r_6), and program functions (r_7, r_8), all sharing the same SDC structure. Each SDC has a (a) Pre-condition: if matching-percentage% of column values in column C, evaluated using a domain evaluation function satisfy inner-distance threshold, we recognize that constraint r should apply to column C, and (b) Post-condition: any value evaluated using the same domain evaluation function satisfy outer-distance threshold, are predicted as data errors. Each example constraint r_i in this table would trigger the detection of a real data error shown in column C_i of Figure 2. All color-coded components (matching-percentage, evaluation function, etc.) are parameters to SDC that are learned using AUTO-TEST.

ID	Type	Pre-condition P: (on what columns should this constraint apply)	Post-condition S: (what values will be predicted as errors)	Conf.
r_1	CTA	85% col vals have their *country-classifier* scores > 0.75	values whose *country-classifier* scores < 0.01	...
r_2	CTA	90% col vals have their *state-classifier* scores > 0.55	values whose *state-classifier* scores < 0.05	...
r_3	Embedding	85% col vals have their *Glove* distances to "january" < 4.0	values whose *Glove* distances to "january" > 5.5	...
r_4	Embedding	80% col vals have their *S-BERT* distances to "seattle" < 1.2	values whose *S-BERT* distances to "seattle" > 1.35	0.88
r'_4	Embedding	90% col vals have their *S-BERT* distances to "seattle" < 1.1	values whose *S-BERT* distances to "seattle" > 1.4	0.93
r_5	Pattern	95% col vals match pattern "\[a-zA-Z]+\d+" (match = 1)	values not matching pattern "\[a-zA-Z]+\d+" (match = 0)	...
r_6	Pattern	95% col vals match pattern "\d+ \[a-zA-Z]+" (match = 1)	values not matching pattern "\d+ \[a-zA-Z]+" (match = 0)	...
r_7	Function	98% col vals return true on function *validate_date()* (ret = 1)	values return false on function *validate_date()* (ret = 0)	...
r_8	Function	99% col vals return true on function *validate_url()* (ret = 1)	values that return false on function *validate_url()* (ret = 0)	...

reliably applied to in a generic fashion to relevant tables, without needing human experts, making them suitable for both "end-user data cleaning" (e.g., in spreadsheets), and "expert-driven data cleaning" (as they serve as a new class of constraints to complement existing constraints).

Intuition: leverage "semantic domain" for error detection. The new class of constraints we study in this work are based on the intuitive notion of "*semantic domains*". Specifically, given any relational table, values in the same column are expected to be *homogeneous* and drawn from a "domain" of same semantics, such as date, url, city-name, address, etc. Figure 2 shows an example table, where the semantics of each column can then be inferred by humans from its values (annotated in column-headers to assist readability).

The semantics of a column often implicitly define the "domain" of valid values that can appear in this column – values falling outside of the "domain" can be picked up by humans as possible data errors, which we show in the example below.

EXAMPLE 1. In Figure 2, we as humans can see that column C_1 is likely a country column given its values, which makes "`Liechstein`" (a misspelling of "`Liechtenstein`") an obvious error in the context of the "semantic domain" (country).

Similarly, most values in column C_2 suggest this to be about state abbreviation codes, which makes "`Germany`" semantically incompatible in the column, and a likely error.

As additional examples, "`febuary`" in C_3 and "`farimont`" in C_4 are clearly misspelled in the context of other values in the columns (which are month-names, and city-names, respectively).

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**C11 (first name)**		**C12 (full name)**		**C13 (city)**		**C14 (gene)**		**C15 (age group)**		**C16 (pay range)**		**C17 (web domain)**
2	aaron		desiree dominguez		whites creek		SOCS4		16-18		Less than $50k		apple.com
3	vicky		hyosik lim		goodlettsville		KIAA0226L		19-24		$50-100K		instagram.com
4	david		hunter smith		old hickory		RP11-6L6.2		25-29		$100-200K		roblox.com
5	omayra		ross rubio		madison		GPSM2		30-34		$200-300K		line.net
6	angie		erik munoz		antioch		ARHGAP27		35-54		$300-500K		google.com.hk
7	mauricio		robin romero		brentwood		RP11-762H8.2		55-64		$500-700K		gstatic.com
8	bruce		nelson jimenez		mount juliet		PRCP		65 & Above		$700-900K		dyndns.info
9	...		...		...		...		...		...		...

Fig. 3. Real examples of table columns, where false-positive detection of errors are produced in highlighted cells, when existing column-type detection techniques are used directly to the task of error-detection.

The values "`fy definition`" in C_5 and "`new facility`" in C_7 are not compatible with other values in the columns and are likely errors (these are likely meta-data as opposed to actual data values).

Finally, the highlighted values in C_6 and C_8 are also inconsistent with the domains implied by other values in the columns (units and urls, respectively), which are therefore likely errors too. □

Given that we as humans can reliably infer column semantics, and then use the underlying "domain of valid values" to identify likely errors in Figure 2 (without needing to understand the domain-specific semantics of a table), the question we ask in this work is whether algorithms can mimic the human intuition, by codifying the intuition into precise and executable data-quality constraints to automatically detect data errors.

"Column-type detection": insufficient for error detection. There is a large literature on the related topic of "column-type detection", where the goal is to predict the semantic type for a given column C, from a pre-defined list of types (e.g., people-names, locations, etc.). The problem has been studied in different settings, leading to techniques tailored to different types of tabular data.

For example, for natural-language-oriented data columns (e.g., people-names, company-names, address, etc.), this is typically studied as a multi-class classification problem, also known as "column type annotation" (CTA) [27, 34, 35, 43, 50, 64, 65, 76, 77], where techniques based on machine learning (ML)-classifiers and text embedding (e.g., Glove [57] and SentenceBERT [60]) are developed.

On the other hand, for machine-generated data columns that are often number-heavy, and with strong regularity (e.g., ip-address, upc-code, time-stamps, etc.), both synthesized and curated regex-patterns / program-functions are used to detect types for such columns [1, 10, 56, 75].

While "column-type detection" is closely related to our goal of using "semantic domain" to detect errors, we observe that applying column-type detection directly to the task of error detection is insufficient, because column-type detection focuses on the *macro-level prediction* of whether a column C belongs to a type T, without being calibrated to make *fine-grained, micro-level predictions* of whether a value-instance $v \in C$ must be an error or not in the context of C and type T. As a result, directly applying column-type detection to error detection can lead to lots of false-positive, like shown below.

Example 2. Figure 3 shows an example table, where column semantics can again be inferred from data values, and are annotated in column headers to assist readability.

Column-type detection methods, such as CTA classifiers and text embedding, can reliably classify column C_{11}, C_{12} and C_{13} as "`first-names`", "`full-names`" and "`city`", respectively. However, when using such classifiers on *individual values* $v \in C$ that happen to be uncommon names, such as "`omayra`", "`hyosik lim`" and "`antioch`", these classifiers will produce low scores, suggesting that the relevant value v may not belong to the type T.

For instance, "`omayra`" (an uncommon name) in C_{11} is not in the vocabulary of Glove embedding [57], and therefore predicted by both embedding and CTA-classifiers to have low-scores for the type "`first-names`". When using these classifiers directly for error-detection, we arrive at the incorrect conclusion that values like "`omayra`" are data errors, which are, in fact, valid (but uncommon) names. The same is true for other uncommon names, such as "`hyosik lim`" in C_{12} and "`antioch`" in C_{13}.

Similarly, in columns C_{14}, C_{15}, C_{16} and C_{17}, a combination of syntactic patterns and NLP-classifiers can be used to recognize the semantics of these columns as "`gene`", "`age-group`", etc. However, just because a value v is in the minority and falls outside of the dominating pattern of a column, does not necessarily make v an error, like the highlighted values in column $C_{14} - C_{17}$ would show, all of which are *not* data errors. Note that these are in contrast to previous examples in $C_5 - C_8$ from Figure 2, where dominating patterns can help to identify both column-types and data errors. □

We can see from the example that column-type detection techniques such as CTA classifiers and patterns cannot be used to detect errors directly, because they are designed to detect semantic types at the column-level (a macro-level prediction), which however are not suited to make fine-grained instance-level predictions for whether a value is erroneous or not (a micro-level prediction).

Auto-Test: Learn reliable "semantic domain constraints". In this work, we propose a new class of data-quality constraints called "Semantic-Domain Constraints (SDC)", that *builds upon and unifies diverse prior techniques for column-type detection*, for our new task of error detection.

Table 1 shows examples of SDC constraints. Briefly, each SDC r consists of a "pre-condition" that tests whether a given column C is in the relevant semantic domain for r to apply, and if so, a "post-condition" would calibrate the confidence of predicting $v \in C$ as an error, mirroring our fuzzy intuition of using "semantic domains" for error detection, but codified in precise constraints. The exact parameters (color-coded components in the table), can be automatically learned from large table corpora using statistical tests in our proposed Auto-Test, which is the focus of this work.

Note that SDC unifies prior techniques for column-type detection, using an abstraction we call "domain evaluation functions" (colored in purple in Table 1), as they can be instantiated using diverse column-type detection, such as CTA-classifiers, text embedding, patterns, and functions, like shown in Table 1.

Extensive evaluations using benchmarks with real spreadsheet and relational tables in the wild, suggest that Auto-Test can reliably learn high-quality SDC constraints for accurate error-detection.

Overall, our proposed Auto-Test has the following key features:

(1) Consistently more accurate than alternative methods, including language models like GPT-4;

(2) Generalizable to new and unseen tables, making it possible to apply SDC with human experts;

(3) Extensible to new column-type detection techniques, all unified in the same framework;

(4) Highly efficient even on large tables, with negligible runtime and memory overhead;

(5) Explainable to humans, as our constraints leverage the natural notion of "semantic domains", which are not black-box models.

2 Related work

Column-type detection. Identifying the semantic types of columns in a table is an important problem, where different techniques are developed that tailor to different types of columns.

For example, for natural-language heavy data content (name, address, etc.), ML-classifiers utilizing embedding features are shown to be effective [27, 34, 35, 64, 76, 77]. For machine-generated data and data with strong regularity (e.g., ip-address, emails, time-stamps, etc.), regex-patterns [1, 29, 36, 54, 63] and program-functions [1, 10, 56, 75] are often used instead. We defer a detailed treatment of existing column-type detection methods to Section 3.

Data Cleaning. Data cleaning is a long-standing challenge in the data management community, with an influential line of research developing constraint-based techniques, including functional dependency (FD), conditional functional dependency (CFD), denial constraints (DC), etc., to detect and repair data errors in tables [13, 19, 28, 39, 46, 61, 74].

While existing data cleaning techniques are both flexible and powerful, they generally rely on complex data-quality constraints (e.g., in first-order logic) to be first defined by experts. Constraint discovery methods have also been studied, though they are generally designed to discover *candidate rules* that still require human experts to verify, in order to ensure accuracy [12, 20, 26, 33, 72]. In contrast, there is an emerging class of "end-user data-cleaning" use cases, especially in spreadsheet software (e.g., Excel and Google Sheets) [4, 6], that aim to empower the masses of non-technical spreadsheet users to clean their data, without the help of experts.

We aim to show that the new class of SDC constraints we introduce, when learned over large table corpora, can apply reliably to new and unseen tables without humans experts, making it applicable to "end-user data-cleaning", while also augmenting existing constraint-based data cleaning by serving as a new and complementary class of constraints.

Outlier detection. There is a large literature of outlier detection methods in the machine learning and data mining literature, as reviewed in [18, 48, 49], which are conceptually related to the problem we study. However, classical outlier detection methods predominantly operate only on *local statistical features* (e.g., value distribution *within a single target column* in our context) to determine outliers, without considering more *global corpus-level information* (e.g., inferred semantic types and global data distributions) that our proposed method specifically leverages for error detection on tabular data. We will show in experiments that this gives our method a unique edge, which substantially outperforms SOTA outlier detection methods from the literature [24].

Language models. Recent advances in NLP show that language models are applicable in a range of table tasks, including data cleaning [42, 53]. Since language models would also understand column semantics, we will empirically compare with state-of-the-art language models like GPT-4.

3 Preliminary: semantic column types

Since our SDC constraints are based on "*semantic types*", we start with an overview of semantic types, and existing techniques to detect them.

Semantic column-type detection methods. As humans, we read tables columns not as string vs. numbers (primitive types), instead, we interpret the semantics of columns (column types), as `date`, `url`, `people-name`, `address`, etc., as shown in Figure 2.

"*Column-type detection*" refers techniques to identify the "semantic types" for a given column C. Diverse techniques have been developed, including ML-classifiers and NL-embedding that are effective for natural-language data (e.g., `people name`, `address`, etc.), and regex-like patterns or program-functions that are suitable for machine-generated data with syntactic structures (e.g., `ip address`, `time-stamps`, etc.).

We survey existing column-type techniques and summarize them into four categories below:
(1) CTA-based methods [35, 64, 76]. In Column Type Annotation (CTA), column-type detection is treated as an ML problem of multi-class classification, that predicts a semantic type from a fixed set of options. Various ML-classifiers have been developed for this problem, such as *Sherlock* [35] classifiers that can detect 78 semantic types ("`type-city`", "`type-country`", etc., from DBPedia), and *Doduo* [64] can further detect 121 semantic types (based on Freebase).

At a conceptual level, a CTA classifier for a semantic type t_i (say "`type-country`")[1], can be viewed as a function f_{cta}, that given a value v (say "`Germany`") as input[2], can produce a classifier score

[1]While some CTA-classifiers such as Sherlock are framed as multi-class classification, they can be equivalently interpreted as multiple binary-classifications (one for each type), to simplify our discussions.
[2]Note that while some CTA-classifiers take an entire column C as input, they also produce valid scores for individual values $v \in C$ (since CTA-classifiers need to make predictions for single-value columns such as $C' = \{v\}$ too).

CTA-classifier(t_i, v) in the range of $[0, 1]$, to indicate the likelihood of v in type t_i, written as:

$$f_{\text{cta}}(t_i, v) = \text{CTA-classifier}(t_i, v)$$

For example, we may get f_{cta}(“`type-country`”, “`Germany`”) = 0.8, and f_{cta}(“`type-city`”, “`Germany`”) = 0.1, from CTA-classifiers.

Observe that $f_{\text{cta}}(t_i, v)$ measures “similarity” between type t_i and value v. To unify CTA with other column-type detection methods, we standardize f_{cta} into a “distance function”, written as f^d_{cta}:

$$f^d_{\text{cta}}(t_i, v) = 1 - f_{\text{cta}} \tag{1}$$

With this distance function, we can equivalently write f^d_{cta}(“`type-country`”, “`Germany`”) = 0.2, and f^d_{cta}(“`type-city`”, “`Germany`”) = 0.9, etc., where a smaller distance indicates a closer association.

(2) Embedding-based methods [57, 60]. Text embedding, such as *Glove* [57] and *SentenceBERT* [60], are popular vector-based representations of text in NLP. In the embedding space, texts with similar semantic meanings (e.g., `month-names` like “`january`”, “`feburary`”, etc.) tend to cluster closely together, while those with unrelated meanings (e.g., “`january`” and `color-names` like “`yellow`”) are positioned further apart [51, 57, 60].

Such embedding provides an effective method to detect semantic types. Specifically, it is natural to select a “*centroid*”, say “`january`”, to represent the semantic-type we want to detect (in this case `month-name`), and for a given a column C, if most or all values $v \in C$ fall within a small radius of “`january`”, we may predict column C as type `month-name` (implied by the centroid “`january`”).

We view text-embedding as providing another function, $f^d_{\text{emb}}(c_i, v)$, that calculates the “distance” between a given value v, and a centroid c_i (representing a semantic-type):

$$f^d_{\text{emb}}(c_i, v) = \text{dist}(\text{emb}(c_i), \text{emb}(v)) \tag{2}$$

For example, let c = “`january`” be a centroid, we may have $f^d_{\text{emb}}(c,$ “`february`”$) = 0.1$, indicating the close proximity of the two values. Alternatively, let c' = “`yellow`” be another centroid (for `color-name`), and we may have $f^d_{\text{emb}}(c',$ “`february`”$) = 0.7$, showing that “`february`” is likely not in the same type as “`yellow`”. Note that f^d_{emb} is already a distance-function, like f^d_{cta} (Equation 1).

(3) Pattern-based methods [29, 36, 54, 63]. For machine-generated data with clear syntactic structures (e.g., `date`, `email`, `timestamp`, etc.), regex-like patterns can often detect semantic types [29, 36, 54, 63]. For example, if most values in C_7 of Figure 2 follow the pattern “`\d{1,2}/\d{1,2}/\d{4}`”, we may predict the column as type `date`.

Similar to CTA and embedding, given a semantic type implied by pattern p_i (e.g., “`\d{1,2}/\d{1,2}/\d{4}`” for `date`), and a value v, we can view the pattern-based detection as a different “similarity” function $f_{\text{pat}}(p_i, v)$ between value v, and a type represented by p_i:

$$f_{\text{pat}}(p_i, v) = \begin{cases} 1, & \text{if } v \text{ matches } p_i \\ 0, & \text{if } v \text{ does not match } p_i \end{cases}$$

We also normalize $f_{\text{pat}}(p_i, v)$ into a distance-function, f^d_{pat}:

$$f^d_{\text{pat}}(p_i, v) = 1 - f_{\text{pat}}(p_i, v) \tag{3}$$

For example, let p = “`\d{1,2}/\d{1,2}/\d{4}`”, v_1 = “`12/3/2020`” and v_2 = “`new facility`” in Figure 2. We have $f^d_{\text{pat}}(p, v_1) = 0$, indicating “distance= 0” between type p and a compatible value v_1; and $f^d_{\text{pat}}(p, v_2) = 1$, indicating “distance = 1” between p and an incompatible value v_2.

(4) Function-based methods [1, 10, 56, 75]. Finally, various “validation-functions” (in python and other languages) have been developed, to validate rich semantic types. For example, `credit-card-number` and `UPC-code` are not just random-numbers, but have internal check-sums

and can be validated using special validation-functions[3] (e.g., Luhn's checksum [2]). Similarly, `date` and `timestamps` can also be validated precisely with functions (in place of simple patterns)[4]. Such "validation functions" are curated in popular open-source repositories like *DataPrep* [56] and *Validators* [10], to reliably detect semantic column types.

For each validation-function f_i (to validate a semantic-type), we similarly view it as a function $f_{\text{fun}}(f_i, v)$, that measures the "similarity" between value v and a type represented by f_i:

$$f_{\text{fun}}(f_i, v) = \begin{cases} 1, & \text{if } f_i(v) \text{ returns true} \\ 0, & \text{if } f_i(v) \text{ returns false} \end{cases}$$

which can again be standardized into a distance-function, f^d_{fun}:

$$f^d_{\text{fun}}(f_i, v) = 1 - f_{\text{fun}}(f_i, v) \tag{4}$$

where a distance $f^d_{\text{fun}}(f_i, v) = 0$ indicates that a value v is validated true by function f_i. For example, let f_i be the *validate_date()* function. Then for C_7 in Figure 2, we have $f^d_{\text{fun}}(f_i, \text{"}\texttt{12/3/2020}\text{"}) = 0$, and $f^d_{\text{fun}}(f_i, \text{"}\texttt{new facility}\text{"}) = 1$.

Observe that different column-type detection methods can have *overlapping coverage* in the types they detect – for example, different CTA-classifiers (e.g., from Sherlock, Doduo, Sato, etc.) all have their own implementations to detect the same semantic type (e.g., `type-city`). Similarly, both pattern-based and function-based methods can detect similar types (e.g., `timestamps`). We do not attempt to manually determine which method is the best for a type t – we simply ingest all type-detection methods into our framework, which can be reasoned consistently to automatically select suitable SDC constraints, which is a salient feature of Auto-Test.

Domain-evaluation function. Note that we intentionally characterize all column-type detection methods as distance functions between value v and a type t (e.g., f^d_{cta}, f^d_{emb}, f^d_{pat}, and f^d_{fun} in Equation (1)-(4)), so that they can be reasoned consistently. Specifically, to quantify whether a value v may be "in" vs. "out of" type t_i for error-detection, we use a notion of "*domain-evaluation functions*" that naturally generalizes these distance-functions.

Definition 1. [**Domain-evaluation function**]. Given a semantic type t_i defined by an underlying column-type detection method (CTA, embedding, etc.), a *domain-evaluation function* $f(t_i, v)$ measures the "distance" between type t_i and value v, where f can be instantiated as f^d_{cta}, f^d_{emb}, f^d_{pat}, and f^d_{fun} in Equation (1)-(4).

As a distance function, a smaller $f(t_i, v)$ naturally indicates that v is likely "in" the domain of type t_i, while a larger $f(t_i, v)$ indicates v to be likely "out" of type t_i.

4 Semantic-Domain Constraints

Given the domain-evaluation functions $f(t_i, v)$ in Section 3, which we will henceforth write as $f_{t_i}(v)$ for simplicity, we now describe a new class of data-quality constraints called Semantic-Domain Constraints (SDC) we propose in this work.

Definition 2. [**Semantic-Domain Constraints**] A Semantic-Domain Constraint (SDC), denoted as $r_t = (P, S, c)$ for semantic type t, is a 3-tuple that consists of a *pre-condition* P, a *post-condition* S, and a *confidence-score* c, where:

[3]For example, https://yozachar.github.io/pyvalidators/stable/api/card/ for "credit-card-number", and https://pypi.org/project/barcodenumber/ for type "UPC-code"
[4]For example, https://docs.dataprep.ai/user_guide/clean/clean_date.html and https://gurkin33.github.io/respect_validation/rules/DateTime/ for "date" and "timestamp".

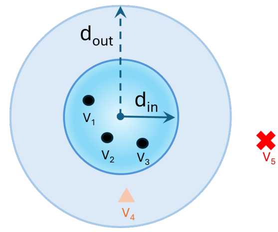


Fig. 4. Visual illustration of a constraint $r_t = (P, S, c)$, where the inner-ball with radius d_{in} corresponds to the pre-condition P, the outer-ball with radius d_{out} corresponds to the post-condition S. For a column $C = \{v_1, v_2, v_3, v_4, v_5\}$, v_1, v_2 and v_3 fall inside the inner-ball (indicating that these values are are likely in the domain of type t), while v_5 falls outside of the outer-ball (and likely not in the type t).

- The pre-condition P: it determines whether the SDC described in r should apply to an input column C, defined as:

$$P(C, f_t, d_{in}, m) = \begin{cases} \text{true} & \text{if } \frac{|\{v|v \in C, f_t(v) \leq d_{in}\}|}{|\{v|v \in C\}|} \geq m, \\ \text{false} & \text{otherwise.} \end{cases}$$

  When the fraction of values $v \in C$ with domain-evaluation function $f_t(v)$ no greater than an *inner-distance threshold* d_{in}, denoted as $\frac{|\{v|v \in C, f_t(v) \leq d_{in}\}|}{|\{v|v \in C\}|}$, is over a *matching-percentage* m, the pre-condition P evaluates true (in which case r_t applies to C), .
- The post-condition S: if the pre-condition P evaluates true, it will be used to detect values $v \in C$ as errors, whose domain-evaluation function $f_t(v)$ evaluates to be greater than an *outer-distance threshold*, d_{out}, written as:

$$S(C, f_t, d_{out}) = \{v|v \in C, f_t(v) > d_{out}\}$$

- The confidence $c \in [0, 1]$: indicates the confidence of the errors detected by the post-condition S.

Intuitively, we can visualize any constraint $r_t = (P, S, c)$ for a type t, as two "concentric balls" depicted in Figure 4, where the pre-condition P corresponds to an inner ball of radius d_{in}, and the post-condition S corresponds an outer ball of radius d_{out}, respectively.

The pre-condition $P(C, f_t, d_{in}, m)$ checks whether a given column C is in the semantic domain of the type t (before r_t can apply). Specifically, it uses the domain-evaluation function $f_t(v)$ for type t, to calculate the fraction of values $v \in C$ that, when evaluated using $f_t(v)$, fall within the inner ball of radius d_{in} (indicating that they belong to type t), written as $\frac{|\{v|v \in C, f_t(v) \leq d_{in}\}|}{|\{v|v \in C\}|}$.

In Figure 4 for instance, for an example column $C = \{v_1, v_2, v_3, v_4, v_5\}$, values v_1, v_2 and v_3 marked in circles fall inside the inner-ball, making this ratio $\frac{3}{5} = 0.6$. If the matching-percentage $m = 0.5$, then C is determined to be in the domain of type t, so that the post-condition S in r_t will apply to C.

The post-condition $S(C, f_t, d_{out})$ would then check whether there are any values $v \in C$ that fall substantially away from the inner ball, to be outside of the outer ball, written as $S = \{v|v \in C, f_t(v) > d_{out}\}$. If such values exist in C, values in S will be predicted as *errors*, with a confidence score c.

In Figure 4 for example, value v_5 marked by a red cross is outside of the outer-ball, and can be predicted as an error. Note that value v_4 marked as a triangle in the figure, falls outside of the inner-ball but inside of the outer-ball, will not be predicted as errors (because intuitively they are not sufficiently farther away from the inner-ball, to be reliably predicted as errors).

Parameters. Note that three parameters, d_{in}, d_{out}, m, are used in each constraint r_t. These are clearly hard to set manually, especially when there are many semantic-types, where each type t

has its own optimal parameters. A key technical challenge in this work is to automatically learn these parameters from real tables, both efficiently and with quality-guarantees (Section 5).

We re-visit the table in Figure 2 to illustrate SDC below.

Example 3. Consider r_4 in Table 1. Its domain-evaluation function f_t is based on *Sentence-BERT* embedding distance, and its type t is implied by the centroid "`seattle`" (of type `city`). The pre-condition P has an inner-ball radius of $d_{in} = 1.2$, meaning values within distance 1.2 to "`seattle`" are believed to be in the same type. Evaluating this r_4 against C_4 in Figure 2, we find 95% of values in C_4 to be within the inner-ball, greater than the required matching-percentage $m = 80\%$, ensuring that r_4 applies to C_4. Checking two values outside of the inner-ball, "`shakopee`" (an uncommon name) and "`farimont`" (a typo), we find the latter to have a distance greater than $d_{out} = 1.35$, therefore falling outside of the outer-ball specified in its post-condition S, suggesting "`farimont`" may be an error with confidence $r_4.c = 0.88$. ("`Shakopee`" falls inside the outer-ball, and is not predicted as an error).

Note that it can be checked that r_4 does not apply to any other columns in Figure 2, as not enough fraction of values in these columns fall in the inner-ball to meet the matching-percentage m requirement (intuitively they are not of type of `city`).

Also note that there exists another constraint r'_4 that has the same centroid "`seattle`", but with a smaller inner-ball radius (1.1), a larger outer-ball radius (1.4), and a larger m (0.9), which intuitively is a stricter and more confident version of r_4, with confidence 0.93. It also triggers on "`farimont`" in C_4, and since r'_4 has a higher confidence than r_4 (0.88), it assigns "`farimont`" a even higher confidence-score. There can be many such variants for the same semantic-type (e.g., r_4 and r'_4) in Table 1, with different parameter configurations, and their corresponding calibrated confidence scores. We assign the confidence of a prediction based on its most confident SDC (e.g., 0.93 for "`farimont`"), which is natural.

Similarly, another constraint r_2 based on CTA-classifiers can detect the incompatible value "`germany`" from C_2. Recall that CTA-classifiers produce similarity-scores, which we standardize into distances (Equation (1)). The pre-condition requires a score greater than 0.55, which would translate to an inner-ball radius of $d_{in} = 0.45$. We find over $m = 90\%$ of C_2 to be in the inner-ball, making r_2 applicable on C_2. Since "`germany`" lays outside of the outer-ball $d_{out} = 1 - 0.05 = 0.95$, making it an error predicted by r_2.

As a final example, r_6 is based on patterns, where match=1 and 0 are transformed into distance of 0 and 1, respectively, using distance functions (Equation (3)), so that we have $d_{in} = 0$ and $d_{out} = 1$, respectively. This r_6 will trigger on column C_6, since over $m = 95\%$ of values match the pattern (distance=0), which fall inside the inner-ball. The only non-matching value "`0.05%`" in C_6 has distance=1, which is outside of the outer-ball and detected as error. □

Why this design of SDC. We design SDC with this structure for the following reasons. First, it mimics the human intuition of identifying data errors – given a table in Figure 2, humans would read values in a column, to first identify its semantic type (e.g., `city` vs. `date`, which is a "pre-condition"), before using a fuzzy notion of "domain" of each type to identify errors (post-conditions). Our design of SDC mimics the reasoning process – by imposing a inner-ball/outer-ball "structure", we constrain the search space of constraints using the strong "prior", effectively reducing the problem to a more tractable form that focuses solely on learning a suitable set of parameters (d_{in}, d_{out}, m).

Because SDC is based on semantic-domains, the resulting constraints are "explainable" as they are often associated with types (e.g., the prediction of "`germany`" in C_2 can be explained using the CTA-classifier for `state`), making predictions interpretable.

Finally, the SDC framework is extensible to different column-type detection methods, and it is easy to add/remove constraints in a white-box fashion, making it easy to deploy and operationalize.

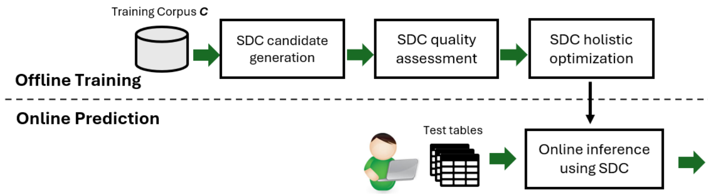


Fig. 5. Architecture diagram of AUTO-TEST

Problem Statement: Learning SDC. In this work, we want to "learn" high-quality SDCs with appropriate parameters from a large table corpus, so that they can cover diverse semantic-types (e.g., in Table 1), and be readily applicable to new and unseen tables.

We leverage a large corpus of tables C (e.g., millions of tables crawled from the web and enterprises), and model them as a collection of individual columns $C = \{C\}$. Ideally, we want to leverage C to learn a set of high-quality SDCs, denoted by R, such that:

(1) *recall* of R is maximized, or R should detect as many errors as possible on unseen test set C_{test};
(2) *false positive rate (FPR)* of R is minimized, for R should trigger few false-positives on C_{test};
(3) *size* of R is not exceedingly large for latency/efficiency reasons (the size of Table 1 is limited).

We give a high-level sketch of our problem below, which will instantiate into concrete problem variants in later sections.

DEFINITION 3. [**Learning Semantic-Domain Constraints**]. Given a corpus C, a size constraint B_{size}, and a FPR threshold B_{FPR}, find a set of SDCs R that maximizes Recall(R), while satisfying $|R| \leq B_{size}$ and $FPR(R) \leq B_{FPR}$, written as:

$$\max_R \text{Recall}(R) \tag{5}$$

$$\text{s.t.} \quad |R| \leq B_{size} \tag{6}$$

$$FPR(R) \leq B_{FPR} \tag{7}$$

Note that in balancing the three requirements, we want to bound FPR (e.g., false-positive rate should not exceed $B_{FPR} = 1\%$, for scenarios like Figure 1 has strict precision requirements), and the size of R (e.g., $|R|$ should not exceed $B_{size} = 10000$) to limit its memory footprint and make inference efficient, while maximizing recall as much as possible.

5 AUTO-TEST: Learn SDC using tables

We now describe our proposed AUTO-TEST that learns high-quality SDCs from a large table corpus C in an unsupervised manner.

Figure 5 shows the architecture of AUTO-TEST, which has an *offline training stage*, where SDCs are learned from given a corpus C, and an *online prediction stage*, where the learned SDCs are applied on new test tables to make prediction. Since online prediction is relatively straightforward and already explained in Example 3, we focus on the offline training part in this section.

The offline training has three steps, which we will describe in turn below. At a high level, we will first generate a set of SDC candidates (Section 5.1), and assess their quality using principled statistical tests (Section 5.2), before we perform holistic optimization of the problem stated in Definition 3, to select an optimal set of SDCs with quality guarantees (Section 5.3).

5.1 SDC Candidate Generation

Our first step in offline training is a preprocessing step that generates a large set of SDCs candidates.

Recall that in Definition 2, each SDC has 4 parameters: domain-evaluation function f_t, inner-distance d_{in}, outer-distance d_{out}, and matching-percentage m (color-coded in Table 1 for readability).

Note that f_t may be instantiated using different "domain-evaluation functions" (Definition 1) for different column-type detection methods, namely f^d_{cta}, f^d_{emb}, f^d_{pat} and f^d_{fun} in Equation (1)-(4), for CTA, embedding, patterns, and functions, respectively. Specifically, we instantiate f_t as follows:

- CTA. We use the 78 classifiers in *Sherlock* [35] (designed to semantic-types in DPBpedia), and the 121 classifiers in *Doduo* [64] (for semantic-types in Freebase), for a total of 199 f^d_{cta} functions.
- Embedding. We use the *Glove* [57] and *SentenceBERT* [60] embedding, and randomly sample 1000 values as centroids (which may be values like "`seattle`" and "`january`" shown in Table 1), to create a total of 2000 f^d_{emb} functions.
- Pattern. We generate common patterns observed in our corpus $\mathbf{C}$, for a total of 45 f^d_{pat} functions.
- Function. We use validation-functions in *DataPrep* [56] and *Validators* [10], as 8 f^d_{fun} functions.

For parameters d_{in}, d_{out}, and m, we perform grid-search and enumerate parameters using fixed step-size (e.g., the matching-percentage m is enumerated with a step-size of 0.05, or $m \in \{1.0, 0.95, 0.9, \ldots\}$, and d_{in}/d_{out} are enumerated similarly). This generates a total of over 100,000 candidate SDCs.

Since these SDC candidates are enumerated in an exhaustive manner, only a small fraction of appropriately parameterized SDCs are suitable for error-detection, which we will identify using (1) statistical tests and (2) principled optimizations, explained below.

5.2 SDC Quality Assessment by Statistical Tests

In this section, we take all SDC candidates, and use statistical hypothesis tests to assess the quality of each candidate r.

Given an SDC $r = (P, S, c)$, where $r.P$ is its pre-condition, $r.S$ its post-condition, and $r.c$ its confidence, from Definition 2. We say a column C is "*covered by*" r, if $r.P(C) = \text{true}$ (i.e., more than m fraction of values in C fall inside the inner-ball with radius d_{in}), in which case C is regarded as "in the semantic domain" specified by the pre-condition $r.P$.

Similarly, we say a column C is "*triggered by*" r, if $r.S(C) = \text{true}$, meaning that the post-condition $r.S$ is producing non-empty results as detected errors in C (i.e., there exists some $v \in C$ that fall outside of the outer-ball with radius d_{out}).

Example 4. We revisit Example 3. Recall that r_4 in Table 1 can be used to detect the error "`farimont`" in C_4 of Figure 2. We say that C_4 is "covered by" r_4, as C_4's semantic domain ("`city`") matches the domain specified in the pre-condition of r_4 (an inner-ball centered at "`seattle`"). No other columns in Figure 2 can be "covered by" r_4 (as they are not columns relating to "`city`", or $r_4.P(C) = \text{false}$).

Because r_4 in Table 1 can detect an error "`farimont`" in C_4, we say C_4 is also "triggered by" r_4 (since $r_4.S(C_4) = \text{true}$). On the other hand, if we remove "`farimont`" from C_4, the resulting column will no longer be "triggered by" r_4, as no more error can be detected. □

Given a large corpus $\mathbf{C}$ and r, we can analyze r's behaviour on $\mathbf{C}$, using a *contingency table* [25, 37] shown in Table 2, where:

- $\left|C^r_{C,T}\right| = \{C | C \in \mathbf{C}, r.P(C) = \text{true}, r.S(C) = \text{true}\}$ denotes the number of columns in $\mathbf{C}$ that are both covered by and triggered by r (C is "in the semantic domain" of r, with errors detected).
- $\left|C^r_{C,\overline{T}}\right| = \{C | C \in \mathbf{C}, r.P(C) = \text{true}, r.S(C) = \text{false}\}$ denotes the the number of columns in $\mathbf{C}$ that are covered by, but not triggered by r (C is "in domain" for r, with no errors detected).

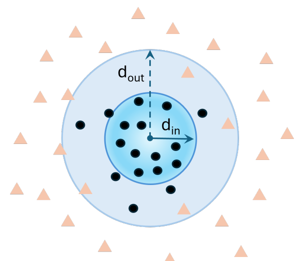


Fig. 6. A visualization of inner/outer-balls that find "natural separation" of semantic-domains, in the universe of all values. Black-dots represent "in-domain" values, triangles represent "out-of-domain" values.

Table 2. Contingency table for an example SDC r, where we perform statistical tests to determine r's efficacy.

	cols covered by r ("in domain" columns)	cols not covered by r ("out of domain" columns)
cols triggered by r (error detected)	$\left\|C^r_{C,T}\right\| = 10$	$\left\|C^r_{\overline{C},T}\right\| = 160,000$
cols not triggered by r (no error detected)	$\left\|C^r_{C,\overline{T}}\right\| = 990$	$\left\|C^r_{\overline{C},\overline{T}}\right\| = 40,000$

- Similarly, $\left|C^r_{\overline{C},T}\right|$ and $\left|C^r_{\overline{C},\overline{T}}\right|$ correspond to the number of columns in C that are not covered by r (C is "not in domain" for r), but with and without detection in the post-condition $r.S(C)$, respectively.

Note that the subscript C and T in these notations would correspond to "cover" and "trigger", respectively. We can see that the top-left entry of Table 2 (covered and triggered), corresponds to set of columns that would be predicted as having errors by r.

Using the contingency table, we perform statistical analysis to: (1) find suitable inner/outer-balls in r that can naturally separate "in-domain" vs. "out-of-domain" columns, and (2) set each r's confidence by the percentage of false-positives it reports among the covered columns. We will explain each in turn below.

(1) Find suitable inner/outer-balls using effect-size (Cohen's h). Recall that in Section 5.1, we exhaustively enumerate SDC candidates with different parameters (inner/outer-ball, centroid, etc.), and the hope is that using an unsupervised analysis of the corpus C, we can identify good SDC that are suitably parameterized.

As we analyze these candidates, we know that a good SDC r for a semantic domain t should ideally have an inner-ball with radius d_{in} tightly enclose most "in-domain" values, and an outer-ball with radius d_{out} that can filter out most "out-of-domain" values, like visualized in Figure 6, where dots and triangles represent in-domain and out-of-domain values, respectively.

More specifically, given a table corpus C, when we compute the contingency table for a candidate r using C, like shown in Table 2, an ideal r with a suitable inner-ball/outer-ball should "cover" a good number of columns in C (e.g., a domain like "`city`" will cover many columns in C), reflected by a large $\left|C^r_{C,T}\right| + \left|C^r_{C,\overline{T}}\right|$ on the left of the contingency table, and at the same time should rarely "trigger" on columns in C, reflected by a small $\left|C^r_{C,T}\right|$ at top-left of the table (r should rarely trigger on columns in C, because the columns we harvested from relational sources are generally clean and error-free – our manual analysis suggests that over 98% columns in C are without errors). In effect, we are looking for r whose ratio $\rho(r) = \left|C^r_{C,T}\right| / \left(\left|C^r_{C,T}\right| + \left|C^r_{C,\overline{T}}\right|\right)$ is small.

In contrast, if the inner/outer-ball are too small or too large (compared to the ideal balls in Figure 6), the separations between "in-domain" vs. "out-of-domain" are no longer clean, and the ratio $\rho(r)$ will be indistinguishable from the same ratio for the vast majority of columns that are "out-of-domain" (the two entries on the right of Table 2), written as: $\overline{\rho}(r) = \left|C^r_{\overline{C},T}\right| / \left(\left|C^r_{\overline{C},T}\right| + \left|C^r_{\overline{C},\overline{T}}\right|\right)$.

Motivated by this observation, we perform statistical tests on $\rho(r)$ and $\overline{\rho}(r)$ in the contingency table (Table 2), to test their *effect size* [38], which is a principled measure of the magnitude of difference between $\rho(r)$ and $\overline{\rho}(r)$ – namely, the larger the difference between the two ratios, the better $\rho(r)$ can "stand out" from the background noise ratio $\overline{\rho}(r)$, indicating a clean in-domain vs. out-of-domain separation using r. We use *Cohen's h* [22] to evaluate the effect size of r, defined as:

$$h(r) = 2\left(\arcsin\sqrt{\rho(r)} - \arcsin\sqrt{\overline{\rho}(r)}\right) \tag{8}$$

Cohen's h has precise statistical interpretations, where $h \geq 0.8$ indicates large effect size [22], which we use to identify SDC candidates with suitably parameterized inner/outer-balls that would correspond to natural domains (e.g., Figure 6).

Example 5. The constraint r_4 in our running example has a contingency table shown in Table 2. It "covers" 1000 columns in C (columns relating to "`cities`"), of which 10 are "triggered" (e.g., Column C_4 in Figure 2). We can calculate $\rho(r_4) = \frac{10}{(990+10)} = 0.01$, which is substantially smaller than the background ratio $\overline{\rho}(r_4) = \frac{160{,}000}{(160{,}000+40{,}000)} = 0.8$. Cohen's h would confirm that r_4 indeed has a large "effect-size" of $h(r_4) = 2.01$. □

Furthermore, we perform "*statistical significance test*" (complementary to effect size in what is known as power-analysis [23]), using the standard Chi-squared tests [58] on the contingency table (Table 2). We discard r whose p value is not significant at 0.05 level.

(2) Estimate r's confidence (Wilson's score intervals). Recall that each SDC $r = (P, S, c)$ has a confidence score c (shown with examples in the last column of Table 1), which is the probability of r not producing false-positives among the "in-domain" columns it covers, that we calibrate using C with precise statistical interpretations as follows.

Specifically, note that the ratio $\hat{c} = \left|C^r_{C,\overline{T}}\right| / \left(\left|C^r_{C,T}\right| + \left|C^r_{C,\overline{T}}\right|\right)$ calculated from our contingency table is exactly an unbiased estimator of c. However, because both $\left|C^r_{C,T}\right|$ and $\left|C^r_{C,\overline{T}}\right|$ (the left two entries of Table 2) can be "rare events" with small counts, whose $\hat{c}$ ratio is therefore susceptible to over- and under-estimation on small samples. In order to guard against this, we produce a "safe" lower-bound of c (as it is better to under-estimate c than over-estimate it, to avoid false-positives), using binomial confidence interval of $\hat{c}$, and specifically we use *Wilson score interval* [71] to produce a lower-bound estimate[5] of the confidence c of a candidate r as:

$$c = 1 - \frac{\left|C^r_{C,T}\right| + \frac{1}{2}z^2}{\left|C^r_C\right| + z^2} - \frac{z}{\left|C^r_C\right| + z^2}\sqrt{\frac{\left|C^r_{C,T}\right| \cdot \left|C^r_{C,\overline{T}}\right|}{\left|C^r_C\right|} + \frac{z^2}{4}} \tag{9}$$

where $\left|C^r_C\right| = \left|C^r_{C,T}\right| + \left|C^r_{C,\overline{T}}\right|$, and $z = 1.65$ is the normal interval width at 95% confidence level.

5.3 SDC Optimizations by LP-Relaxation

Let R_{all} be the set of all candidate SDCs that meet the statistical tests performed in Section 5.2 (still a large set in tens of thousands, with overlapping coverage and varying degrees of quality). We

[5] Note that because the corpus C is not perfectly clean, the true number of false-triggers is bound to be smaller than the current estimate of $\left|C^r_{C,T}\right|$ using C. However, because this is in the denominator of $\rho(r)$, it does not affect our lower-bound analysis, as an over-estimate of $\left|C^r_{C,T}\right|$ still leads to a conservative lower-bound of c, which is what we want.

now describe the key final step in offline-training, where we perform holistic optimization like sketched in Definition 3, to select an optimal set $R \subseteq R_{all}$ with FPR and recall guarantees.

We will first describe how to estimate FPR(r) and recall(r) below.

Estimating FPR. Recall that the FPR of a constraint r is defined as $\text{FPR}(r) = \frac{\text{r-false-positive-columns}}{\text{total-negative-columns}}$, or the number of false-positives r produces, over the total number of negative (error-free) columns. While we don't have labeled data to count these events for r precisely (which would be hugely expensive if we were to label each r), we can approximate these events using a large corpus $\mathbf{C}$ in an unsupervised data-driven manner.

Specifically, since $\mathbf{C}$ is extracted from relational sources, its columns are generally clean and free of errors (for instance, in our manual analysis of a sample of 2400 table columns randomly sampled from spreadsheets and relational tables, we found the error rate of $\mathbf{C}$ to be around 2%, like we will explain in Section 6.1). We can therefore use $|\mathbf{C}|$ to approximate the number of total negative columns in $|\mathbf{C}|$. Also recall that we can estimate false-positives of r, based on $\mathbf{C}^r_{C,T}$ in the contingency table (estimated using $\mathbf{C}$), so putting the two together we can then estimate FPR(r) as $\frac{|\mathbf{C}^r_{C,T}|}{|\mathbf{C}|}$.

Estimating recall. The recall of r, written as Recall(r), is the total number of true-positive errors that r can detect.[6] Since we also don't have labeled data to estimate recall for each r, we use *distant-supervision* [32, 52, 66] to approximate it.

Specifically, we construct a synthetic corpus for that purpose, written as $\mathbf{C}_{syn} = \{C(v^e) = C \cup \{v^e\} | C \in \mathbf{C}, C' \in \mathbf{C}, v^e \in C'\}$, where each column in $\mathbf{C}_{syn}$ is constructed as $C(v^e) = C \cup \{v^e\}$, with C being a randomly sampled column in $\mathbf{C}$, v^e being a randomly sampled value from a different column C', such that when v^e is inserted into C to produce $C(v^e)$, v^e is likely an error in the context of $C(v^e)$. Like in distant supervision [32, 52, 66], this then allows us to compute the set of errors that r can detect in $\mathbf{C}_{syn}$, as

$$D(r) = \{C(v^e) | C(v^e) \in \mathbf{C}_{syn}, r(C(v^e)) = v^e\} \tag{10}$$

where $r(C(v^e)) = v^e$ indicates that r can detect the same v^e as constructed in column $C(v^e)$. We then simply use Recall(r) = $|D(r)|$, as the estimated recall of r.

Coarse-grained SDC Selection (CSS). We are now ready to instantiate the high-level problem sketched in Definition 3 as follows.

Definition 4. **[Coarse-grained SDC Selection (CSS)]**. Given all SDC candidates R_{all}, find a set $R \subseteq R_{all}$ such that its recall Recall(R) is maximized, subject to a constraint that the FPR(R) should not exceed B_{FPR}, and a cardinality constraint that the size of R should not exceed B_{size}, written as:

$$\text{(CSS)} \qquad \max_{R \subseteq R_{all}} \left| \bigcup_{r \in R} D(r) \right| \tag{11}$$

$$\text{s.t. } |R| \leq B_{size} \tag{12}$$

$$\sum_{r \in R} \text{FPR}(r) \leq B_{FPR} \tag{13}$$

Note that in the objective function Equation (11), we use Recall(R) = $|\bigcup_{r \in R} D(r)|$ to instantiate the objective function of the original problem in Definition 3 (Equation (5)), since Recall(R) over a set of constraints R can be calculated as the union of errors detected by each $r \in R$.

Observe that because individual $r \in R_{all}$ can often have overlapping coverage (e.g., different embedding methods, and different CTA-classifiers that can detect columns of a type, say "`city`",

[6] We use the absolute version of recall over the relative version for its simplicity, the two versions differ only by a universal denominator (the total-number-of-positive-columns), and are therefore equivalent in our context.

Algorithm 1: COARSE-SELECT

input : All candidate SDC R_{all}
output : The selected set $R \subseteq R_{all}$

1: Transform a CSS problem instance into a CSS-ILP instance
2: Transform a CSS-ILP instance into a LP-relaxation version CSS-LP
3: $\{x_i\} \leftarrow$ optimal solutions to CSS-LP, solved using LP solvers
4: $R \leftarrow \{\}$
5: **for** $r_i \in R_{all}$ **do**
6: $\quad R \leftarrow R \cup \{r_i\}$, with probability x_i
7: **end for**
8: **return** R

are all present in R_{all}), this union term in Equation (11) can therefore take the overlaps of recall into consideration when we optimize for the best solution set R.

Also note that in Equation (13), we use $\sum_{r \in R} \mathrm{FPR}(r)$ in place of $\mathrm{FPR}(R)$ in Equation (7) of Definition 3, because it can be verified that $\mathrm{FPR}(R) \leq \sum_{r \in R} \mathrm{FPR}(r)$ (using an argument similar to union-bound), so that imposing the constraint in Equation (13) esures that the original constraint $\mathrm{FPR}(R) \leq B_{FPR}$ is also satisfied.

We show that the CSS problem in Definition 4 is hard and hard to approximate, using a reduction from maximum coverage. A proof of this can be found in [5].

THEOREM 5.1. *The CSS problem is NP-hard and cannot be approximated with a factor of* $(1 - 1/e)$, *unless* $NP \subseteq DTIME(n^{O(\log \log n)})$.

Despite its hardness, we develop COARSE-SELECT in Algorithm 1 to solve CSS using LP-relaxation and randomized rounding [59], which has an approximation ratio of $(1 - 1/e)$ (matching the inapproximability result). Specifically, we first transform CSS into a CSS-ILP problem:

$$\text{(CSS-ILP) maximize} \sum_{C_j \in C_{syn}} y_j \tag{14}$$

$$\text{s.t.} \sum_{r_i \in R_{all}} x_i \leq B_{size} \tag{15}$$

$$\sum_{r_i \in R_{all}} \mathrm{FPR}(r_i) \cdot x_i \leq B_{FPR} \tag{16}$$

$$\sum_{r_i \in K_j} x_i \geq y_j \quad \forall C_j \in C_{syn} \tag{17}$$

$$x_i, y_j \in \{0, 1\} \tag{18}$$

Here, we use an indicator variable x_i for each $r_i \in R_{all}$, where $x_i = 1$ indicates r_i is selected into R, and 0 otherwise. Let $D(R) = \bigcup_{r \in R} D(r)$ be the union of all errors detected by R. We use another indicator variable y_j for each column $C_j \in C_{syn}$, where $y_j = 1$ indicates $C_j \in D(R)$, and 0 otherwise. Finally, for each $C_j \in C_{syn}$, we define $K_j \subseteq R_{all}$ as the set of constraints that can detect the error constructed in C_j. It can be shown that the CSS-ILP problem so constructed, has the same solution as the original CSS problem.

From CSS-ILP, we construct its LP-relaxation [59], referred to as CSS-LP, by dropping its integrality constraint in Equation (18). The resulting CSS-LP is a linear program that can be solved optimally in polynomial-time, yielding *fractional solution* for each x_i. Finally, we use randomized-rounding like shown at the end of Algorithm 1, to turn the fractional x_i into integral solutions R.

We show a proof in [5] that the solution from Algorithm 1 provides the following guarantees.

Theorem 5.2. *Let R be the solution returned by Algorithm 1, and $E(\cdot)$ denote expectation. Then the following hold: $E(|R|) \leq B_{size}$, $E(\sum_{r \in R} FPR(r)) \leq B_{FPR}$, and $E(|D(R)|) \geq (1 - 1/e)OPT$ where OPT is the optimal value.*

Note that in expectation, our approximation ratio matches the inapproximability in Theorem 5.1.

Fine-grained SDC Selection (FSS). While CSS reduces R_{all} into R from the perspective of set-based optimization, we find in our evaluation, that the confidence produced by the solution $R \subseteq R_{all}$ from CSS, to deviate substantially from the true calibrated confidence (Equation (9) in Section 5.2), if we use the entire R_{all}. This is because CSS only focuses on the set-based optimization, without considering how well the selected R can approximate the calibrated confidence from the original R_{all}, which leads to poor confidence ranking of predicted results, that negatively affects the result quality (e.g., when evaluated using area under precision-recall curves).

To address this inadequacy, we propose an improved version of CSS that ensures confidence approximation in the selection process, which we call *Fine-grained SDC Selection (FSS).* Define $\text{diff}(C, R, R_{all}) = \text{conf}(C, R_{all}) - \text{conf}(C, R)$ as the difference in predicted confidence for any $C \in C_{syn}$, between using R_{all} and R. We define FSS as follows:

Definition 5. [**Fine-grained SDC Selection (FSS)**]. Given all SDC candidates R_{all}, find a set $R \subseteq R_{all}$ to maximize the number of columns detected in $D(R)$, whose predicted confidence using R does not deviate from its true confidence by δ (or $\text{diff}(C, R, R_{all}) \leq \delta$), subject to a constraint that the FPR(R) should not exceed B_{FPR}, and the size of R should not exceed B_{size}, written as:

$$\text{(FSS)} \qquad \max_{R \subseteq R_{all}} |\{C \mid C \in D(R), \text{diff}(C, R, R_{all}) \leq \delta\}|$$
$$\text{s.t.} \ |R| \leq B_{size}$$
$$\sum_{r \in R} \text{FPR}(r) \leq B_{FPR}$$

Observe that when we set $\delta = 1$, FSS reduces to CSS because the confidence approximation requirement of $\text{diff}(C, R, R_{all}) \leq \delta$ is trivially satisfied, making it an advanced variant of CSS.

We propose algorithm Fine-Select to solve FSS, also with quality guarantees. The pseudo-code for Fine-Select is similar to that of Coarse-Select (Algorithm 1) with two key modifications: (1) for each column $C_j \in C_{syn}$, its indicator variable $y_j = 1$ if $C_j \in \{C \in D(R) \mid \text{diff}(C, R, R_{all}) \leq \delta\}$, and 0 otherwise; (2) for each $C_j \in C_{syn}$, we set $K_j \subseteq R_{all}$ as the set that can detect the error constructed in C_j, with the required confidence approximation specified by δ.

We show in our technical report [5] that the Fine-Select approach has a $(1 - 1/e)$ approximation ratio in expectation, with all constraints also satisfied in expectation, like stated below.

Theorem 5.3. *Let $R \subseteq R_{all}$ be the solution produced by Fine-Select, and $E(\cdot)$ denote expectation. Then the following hold: $E(|R|) \leq B_{size}$, $E(\sum_{r \in R} FPR(r)) \leq B_{FPR}$, and $E(|\{C \in D(R) \mid diff(C, R, R_{all}) \leq \delta\}|) \geq (1 - 1/e)OPT$ where OPT is the optimal value.*

We compare Fine-Select and Coarse-Select for their effectiveness in our experiments.

6 Experiment

We perform extensive evaluations using real errors from real data. Our code, data, and labeled benchmarks are available for future research[7].

[7] https://github.com/qixuchen/AutoTest

6.1 Experimental Setup

Benchmarks. To test the effectiveness of different algorithms on real tables "in the wild", we focus on real relational tables and spreadsheet tables in our evaluation, and create two benchmarks containing real errors from real tables, described below.

Spreadsheet-Table-Bench (ST-Bench). We randomly sample 1200 spreadsheet columns, extracted from real spreadsheets (`.xlsx` files crawled from the web), as our ST-Bench test set.[8] Each column is carefully labelled and cross-checked by human-labellers, as either *clean* (no data errors are present); or *dirty*, in which case all erroneous values in the column are marked in ground-truth for evaluation. A total of 47 columns (3.9%) contain real errors.

Relational-Table-Bench (RT-Bench). We also sample 1200 relational table columns from real tables extracted from BI models (`.pbix` files crawled from the web), as our second test set. Each column is similarly labeled as clean, or dirty, with erroneous values identified in the ground-truth. A total of 40 columns (3.3%) are identified to contain errors.

Existing data-cleaning benchmarks. To test the applicability of learned SDCs on existing data-cleaning benchmarks, we further compile 9 commonly-used datasets from prior studies [30, 46, 47, 55], which are: *adults*, *beers*, *flights*, *food*, *hosptial*, *movies*, *rayyan*, *soccer* and *tax*. We reuse existing ground-truth in our evaluation.

Evaluation metrics. We evaluate the quality of all algorithms, using standard precision/recall, where precision $P = \frac{\text{num-of-correct-predicted-errors}}{\text{num-of-total-predicted-errors}}$, and recall $R = \frac{\text{num-of-correct-predicted-errors}}{\text{num-of-total-true-errors}}$.

Since each algorithm have different score thresholds to make predictions at different confidence levels, we plot precision-recall curves (PR-Curves) of all algorithms, and summarize the overall quality of PR-Curves using two standard metrics:

(1) Precision-Recall Area-Under-Curve (PR-AUC) [11], which measures the area under the PR-curve, where higher is better;

(2) F1-score at Precision=0.8 (F1@P=0.8) [14], which measures the F1 score (the harmonic mean of precision and recall) at high precision ($P = 0.8$). Note that a high level of precision is crucial in our setting (e.g., to win user trust in end-user data cleaning), which is why we use this metric to complement PR-AUC.

6.2 Methods Compared

We compare with the following methods on quality and efficiency.

- **Column-type detection methods**. Our first group of baselines directly invoke existing column-type detection methods. For each method below, we compute the domain evaluation score $f_t(v)$ of type t (Definition 1), for each value v in column C, and use the standard z-score on the resulting distribution of $f_t(v)$ to identify potential errors [31]. We vary the z-score threshold to plot PR-curves for each method.
  - CTA methods: *Sherlock* [35], *Doduo* [64]. We use the CTA-classifier as domain evaluation function $f_t(\cdot)$ to compute a score distribution for each column C.
  - Embedding domains: *Glove* [57], *Sentence-BERT* [60]. We use the embedding distance as the domain evaluation function $f_t(\cdot)$ to compute a score distribution.
  - Function domains: *DataPrep* [1], *Validator* [10]. We use the boolean result returned by a type-validation function (true=1, false=0) as the domain evaluation function $f_t(\cdot)$.
  - Pattern domains: *Regex*. We use whether a value v matches an inferred regex pattern of a column C (match = 1, non-match = 0) as the domain evaluation function $f_t(\cdot)$.

[8] We sample non-numerical columns for testing only, since it is usually trivial to identify non-conforming values (e.g., strings) in numerical columns.

- **GPT-4**. Language models such as GPT have shown strong abilities in diverse tasks [16]. Since our task can also be formulated as a natural-language task, we invoke GPT-4[9] with extensive prompt optimization, including (1) select few-shot examples [16]; and (2) use chain of thought (COT) [70] (to require GPT-4 to reason about its detection with possible repairs, so that it can stay truthful with fewer false-positives). We report 4 variants here based on prompts used, which are: few-shot-with-COT, few-shot-no-COT, zero-shot-with-COT, and zero-shot-no-COT.
- **Katara** [21]. Katara performs data cleaning by mapping table columns to Knowledge-Bases (KB) like YAGO, to identify columns of type `city`, `country`, etc., so that errors can be detected. This is similar in spirit to ours, but is limited to symbolic knowledge-bases, and are based on heuristic mapping with static thresholds (not trained/calibrated).
- **Auto-Detect** [32]. This approach detects errors due to incompatible data patterns, based on co-occurrence statistics. While it also leverages a corpus to produce predictions, it is only applicable to patterns, limiting its coverage.
- **Outlier detection methods**. There is a large literature on outlier detection, we select *RKDE* [40], *PPCA* [68] and *IForest* [45] for comparison, which are shown to be the best-performed methods in an empirical study [24]. We also include three classical methods: *SVDD* [67], *DBOD* [41] and *LOF* [15], that compared with in an earlier study [32] similar to our problem setting.
- **Commercial**. We also test our benchmarks on two commercial software targeting non-technical end-users, that can automatically detect errors in tables. We refer to these two systems as Vendor-A and Vendor-B in our experiments.
- **Auto-Test**. This is our proposed method. We report 3 variants of Auto-Test, which are (1) All-Constraints, which uses the entire set of candidate constraints R_{all} after quality-based pruning (Section 5.2), (2) Coarse-Select (Algorithm 1), and (3) Fine-Select (Section 5.3).
  We invoke the solver in SciPy [62] to solve our LP. By default, we set B_{size} to 500, B_{FPR} to 0.1, and δ in Fine-Select to 10^{-3}.
  For training, we use three corpora: (i) 247K relational table columns extracted from relatoinal sources [44], henceforth referred to as Relational-Tables, and (ii) 297K spreadsheet table columns extracted from real spreadsheets, referred to as Spreadsheet-Tables, and (iii) 298K real table columns extracted from a publically available corpus Tablib [9]. We kept Relational-Tables and Spreadsheet-Tables completely separate from RT-Bench and ST-Bench. To test generalizability, we also train on one corpus (e.g., Relational-Tables) and test on the benchmark from a different source (e.g., ST-Bench).

All experiments are run on a Linux machine with a 64-core, 2.4 GHz CPU and 512 GB memory.

6.3 Quality Comparisons

Quality comparison with real errors. In Figure 7 and 8, we compare the PR-curves of all methods, on two benchmarks Relational-Table-Bench (RT-Bench) and Spreadsheet-Table-Bench (ST-Bench), respectively. To avoid clutter in these figures, we show the best method Fine-Select from the Auto-Test family, trained using Relational-Tables (additional results can be found in our full technical report [5]). Similarly, for methods in GPT-4 family, we also only show few-shot-with-COT since it performs the best, as can be seen in Table 3.

The proposed Fine-Select substantially outperforms all other methods. It is worth noting that Fine-Select trained using Relational-Tables not only performs well on RT-Bench, but also on ST-Bench, demonstrating strong generalizability (spreadsheet vs. relational tables).

Among all baselines, SentenceBERT, DataPrep, and Regex perform better than other domain-based baselines, while RKDE performs better than other outlier detection baselines, but these

[9]Version gpt-4-0125, accessed via OpenAI API in 2024-06.

Table 3. Quality comparisons, reported as (F1@P=0.8, and PR-AUC), on ST-Bench and RT-Bench.

	Method	Spreadsheet-Table-Bench (ST-Bench) real	+5% syn err.	+ 10% syn err.	+20% syn err.	Relational-Table-Bench (RT-Bench) real	+5% syn err.	+10% syn err.	+20% syn err.
Ours	All-Constraints	0.23, 0.38	0.36, 0.39	0.47, 0.57	0.50, 0.66	**0.21, 0.34**	**0.30**, 0.36	**0.36**, 0.48	0.36, 0.54
	Fine-Select	**0.34, 0.45**	**0.38, 0.52**	**0.48, 0.62**	**0.53, 0.68**	**0.21, 0.34**	**0.30, 0.46**	**0.36, 0.56**	**0.40, 0.62**
	Coarse-Select	0.25, 0.43	0.35, 0.52	0.41, 0.60	0.52, 0.67	0.05, 0.31	0.25, 0.43	0.28, 0.53	0.39, 0.61
Column-type detection methods	Sherlock	0, 0.04	0, 0.05	0, 0.10	0.01, 0.21	0, 0.03	0, 0.06	0, 0.14	0, 0.22
	Doduo	0.04, 0.06	0.06, 0.09	0.09, 0.17	0.08, 0.31	0, 0.03	0, 0.05	0, 0.10	0, 0.20
	Glove	0, 0.10	0.03, 0.18	0.07, 0.26	0.06, 0.35	0.05, 0.10	0.06, 0.13	0.03, 0.18	0.03, 0.28
	SentenceBERT	0.08, 0.14	0.12, 0.18	0.11, 0.23	0.18, 0.36	0.09, 0.09	0.14, 0.18	0.11, 0.19	0.09, 0.28
	Regex	0.04, 0.25	0.06, 0.30	0.09, 0.41	0.27, 0.51	0, 0.14	0.03, 0.28	0.01, 0.38	0.11, 0.48
	DataPrep	0.08, 0.22	0.09, 0.25	0.10, 0.38	0.12, 0.49	0.05, 0.14	0.06, 0.24	0.03, 0.40	0.03, 0.50
	Validators	0.04, 0.29	0.03, 0.29	0.01, 0.31	0.01, 0.44	0, 0.03	0, 0.05	0, 0.30	0.03, 0.44
Data-cleaning	AutoDetect	0, 0.18	0, 0.17	0, 0.18	0, 0.25	0, 0.09	0, 0.12	0, 0.15	0.01, 0.25
	Katara	0, 0.04	0, 0.05	0, 0.10	0, 0.20	0, 0.03	0, 0.05	0, 0.10	0, 0.19
Outlier detection methods	SVDD	0.04, 0.04	0.06, 0.06	0.09, 0.10	0.09, 0.15	0.05, 0.04	0.06, 0.06	0.03, 0.07	0.03, 0.12
	DBOD	0, 0.15	0, 0.23	0, 0.35	0, 0.46	0, 0.12	0, 0.29	0, 0.40	0, 0.51
	LOF	0, 0.08	0, 0.12	0, 0.18	0, 0.24	0, 0.04	0, 0.12	0, 0.16	0, 0.22
	RKDE	0.04, 0.20	0.06, 0.24	0.09, 0.31	0.24, 0.40	0.05, 0.11	0.06, 0.21	0.03, 0.27	0.12, 0.35
	PPCA	0, 0.14	0, 0.15	0, 0.19	0.17, 0.26	0, 0.06	0, 0.12	0, 0.15	0, 0.20
	IForest	0, 0.13	0, 0.15	0, 0.19	0.11, 0.25	0, 0.05	0, 0.13	0.11, 0.17	0.12, 0.22
GPT	few-shot-with-COT	0, 0.20	0, 0.30	0, 0.38	0, 0.56	0, 0.16	0, 0.33	0, 0.48	0, 0.53
	few-shot-no-COT	0, 0.20	0, 0.32	0, 0.38	0, 0.56	0, 0.10	0, 0.22	0, 0.44	0, 0.56
	zero-shot-with-COT	0, 0.15	0, 0.28	0, 0.34	0, 0.53	0, 0.16	0, 0.26	0, 0.43	0, 0.52
	zero-shot-no-COT	0, 0.11	0, 0.23	0, 0.25	0, 0.43	0, 0.08	0, 0.21	0, 0.40	0, 0.46
Commercial	Vendor-A	0, 0.18	0, 0.20	0, 0.22	0, 0.27	0, 0.02	0, 0.05	0, 0.11	0, 0.21
	Vendor-B	0, 0.02	0, 0.05	0, 0.10	0, 0.21	0, 0.02	0, 0.05	0, 0.11	0, 0.21

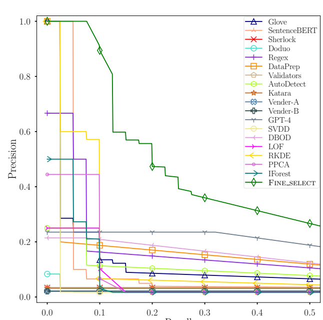


Fig. 7. PR curves on the 1200-column RT-Bench, trained on Relational-Tables.

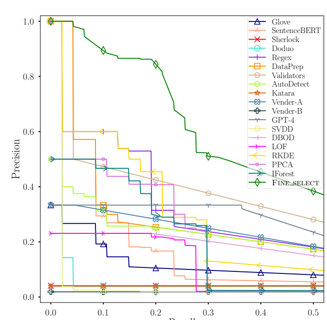


Fig. 8. PR curves on the 1200-column ST-Bench, trained on Relational-Tables.

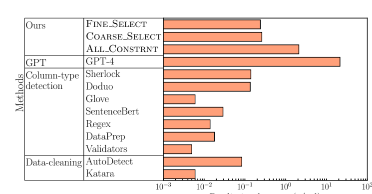


Fig. 9. Online prediction latency: average time to process one column (all methods)

methods still lag significantly behind the proposed Fine-Select. Note that while GPT-4 can detect many data errors (around 80%), it also produce a large number of false-positives (especially on columns involving code-names, abbreviations, and proprietary vocabularies that are not standard English), which affects its quality.

In Table 3, we further summarize the PR-curves using two metric numbers: (1) F1@P=0.8, and (2) PR-AUC, both of which show a picture similar to what we observe on the PR-curves, where Fine-Select outperforms alternatives methods.

Quality comparison with real and synthetic errors. In addition to testing on the real RT-Bench and ST-Bench, we further report 3 settings for each of the benchmark in Table 3, where we inject synthetic errors (using values randomly sampled from other columns), at 5%/10%/20% levels, on top of real errors. We observe that Fine-Select continues to dominate all other methods, confirming its effectiveness across a spectrum of error rates.

Coverage of specialized content. In addition to precision/recall, a question we want to explore is whether the generated SDCs only cover well-known concepts commonly represented on the web. Figure 10 shows a sampled analysis, with examples of real columns that have specialized meanings and structures (corresponding to contract-number, article-number, etc.), some of which are likely

col_header	data_vals
movie_id	['tt0054215', 'tt0088993', 'tt0032484', 'tt0889671', 'tt1325014', 'tt0065112', 'tt0074512', 'tt0092796', 'tt0090021', 'tt0102733', 'tt0102741', 'tt1045642', 'tt0265666', 'tt0440963', 'tt
contract no.	['b50005237', 'b50005238', 'b50002613', 'b50003343', 'b50004853', 'b50003298', 'b50003292', 'b50004499', 'b50004017', 'b50003446', 'b50004085', 'b50002318', 'b50004162', 'b
article number	['1-15-80-00-30-spp', '1-20-10-60-00-pek', '1-20-10-62-00-pek', '1-25-50-10-00-kvs', '1-25-50-32-00-ppv', '1-25-50-32-00-wiv', '1-30-50-10-00-kst', '1-30-51-00-00-ksv', '1-40-10-0
commbuys mbpo link	['po-17-1080-osd03-osd03-10332', 'po-17-1080-osd03-osd03-10324', 'po-17-1080-osd03-osd03-10816', 'po-17-1080-osd03-osd03-10336', 'po-17-1080-osd03-src02-10355', 'po-
ordernum	['num000002', 'num000003', 'num000004', 'num000005', 'num000006', 'num000007', 'num000008', 'num000009', 'num000010', 'num000011', 'num000012', 'num000013', 'num
primary funding agreement	['12-ec-002', '13-hqjdf-004', '13-hqjtc-018', '12-100-004', '14-hqjdf-025', '12-130-021', '13-hqjdf-003', '12-ez-016', '14-ez-007', '12-ez-012', '14-hqjdf-009', '13-ez-013', '14-hqjdf-0

Fig. 10. Examples columns with specialized meanings, but are still "covered" by SDCs: each row here corresponds to a real data column, with its column-header and data-values listed. Many of these columns convey specialized meanings (e.g., specialized contract no., article number, etc.), which are nevertheless covered by our pattern-based SDCs, as our method learns a generalized notation of what a reliable pattern-domain may look like, which transcends specific meanings in each column.

Table 4. Quality and latency comparison for Fine-Select, as we vary the constraint count budget (B_{size})

	ST-Bench					RT-Bench				
Constraint count budget (B_{size})	100	200	500	1000	All-Constraints (26673)	100	200	500	1000	All-Constraints (26673)
Quality: F1@P=0.8	0.29	0.31	0.34	**0.35**	0.23	0.09	0.11	**0.21**	**0.21**	**0.21**
Quality: PR-AUC	0.42	0.41	0.44	**0.46**	0.38	0.22	0.27	**0.34**	0.31	**0.34**
Latency: second per column	0.13	0.16	0.21	0.23	1.44	0.12	0.18	0.24	0.26	2.10

Table 5. Sensitivity to different training corpora

	Spreadsheet-Table-Bench (ST-Bench)				Relational-Table-Bench (RT-Bench)			
	real	+5% syn err.	+ 10% syn err.	+20% syn err.	real	+5% syn err.	+10% syn err.	+20% syn err.
Relational-Tables	0.34, 0.45	0.38, 0.52	0.48, 0.62	0.53, 0.68	0.21, 0.34	0.30, 0.46	0.36, 0.56	0.40, 0.62
Spreadsheet-Tables	0.05, 0.30	0.18, 0.43	0.28, 0.52	0.45, 0.64	0.02, 0.29	0.25, 0.43	0.25, 0.47	0.27, 0.55
Tablib	0.15, 0.45	0.34, 0.54	0.45, 0.61	0.53, 0.68	0.13, 0.41	0.37, 0.54	0.40, 0.56	0.46, 0.60

unique to a specialize domain or few datasets. Our pattern-based SDCs can nevertheless reliably install pattern-based SDCs for such columns, as the method learns a generalized notation of what a reliable domain-pattern should look like, which transcends specific meanings conveyed in the data, therefore providing "coverage" even when the underlying domains may be highly specialized.

We show additional results, e.g., training using different corpora, in [5] in the interest of space.

6.4 Efficiency Analysis

Online prediction latency. Figure 9 shows the average latency of making predictions for one column. The proposed Fine-Select takes around 0.2 seconds on average, which is interactive and suitable for user-in-the-loop scenarios. All-Constraints in comparison, is an order of magnitude slower, showing the benefit Fine-Select in compressing and selecting most beneficial SDCs. GPT-4 is the slowest as it takes over 20 seconds for one column on average.

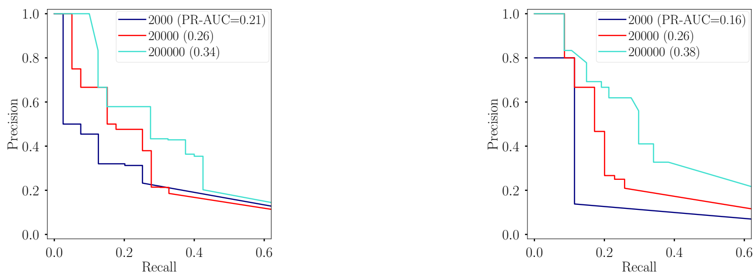
Additional results on latency, including offline latency analysis, can be found in [5].

6.5 Sensitivity Analysis

We analyze the sensitivity of Auto-Test to different parameters.

Sensitivity to the number of constraints. Table 4 shows the effect of varying the number of constraints B_{size} in Fine-Select, using All-Constraints (with 26673 constraints) and GPT-4 as reference points. Fine-Select shows strong efficiency benefit (7-10x faster) over All-Constraints, while having the same or even better quality with just 500 constraints (e.g., Fine-Select shows even higher PR-AUC and F1@P=0.8 than All-Constraints on RT-Bench, likely because it is forced to select high-quality SDCs).

Sensitivity to training corpora. We summarize the performance of using Relational-Tables, Spreadsheet-Tables and Tablib as the training corpus in Table 5. Our results show that the performance with Relational-Tables and Tablib follows a similar trend, both are better than Spreadsheet-Tables. This can be attributed to the fact that Spreadsheet-Tables are crawled from human-generated spreadsheet tables, which tend to be noisier than the machine-generated



(a) RT-Bench (b) ST-Bench

Fig. 11. Sensitivity to training corpus size

Table 6. Quality results of applying SDCs learned in Auto-Test on existing data-cleaning benchmarks. Note that the cell-level precision (reported in the last line) is evaluated by strictly comparing our detection, vs. the ground-truth "clean" version of the benchmarks, which however underestimates the true precision, because the current ground-truth labels in some of the benchmarks are incomplete that "miss" real errors. We therefore also report adjusted precision in parenthesis "()" – for example, our 9-dataset aggregate precision is "95% (97%)", meaning that precision is 95% (174/183) if strictly using existing ground-truth, which however increases to 97% (179/183) if we use augmented ground-truth that is manually labeled and shared at [5].

		9-dataset overall	*adults*	*beers*	*flights*	*food*	*hosptial*	*movies*	*rayyan*	*soccer*	*tax*
Dataset statistics	# of total categorical cols	85	9	6	6	10	16	14	8	8	8
	# of cols covered by existing ground-truth	36	1	3	4	1	12	0	8	1	6
Quality: column-level	Coverage: # of cols with new constraints by using SDC	**17**	2	2	0	3	4	2	1	2	1
	Precision: % of new SDCs that are correct	**94%**	100%	67%	-	100%	100%	100%	100%	100%	100%
Quality: cell-level	True-positives: # of detected data errors using SDCs	**183**	0	5	0	3	13	161	1	0	0
	Precision: % of detected data errors that are correct	**95% (97%)**	-	40% (60%)	-	0% (33%)	92% (100%)	99% (100%)	0% (100%)	-	-

tables in Relational-Tables and Tablib. This observation suggests that the quality of the training corpus plays a critical role in the effectiveness of mined SDCs.

Sensitivity to training corpus size. We study the effect of varying training corpus size from 2,000 to 200,000 columns in Figure 11. On both benchmarks, quality improves with more training data, showing the effectiveness of our data-driven approach.

Robustness to low-quality SDC candidates. To test whether Auto-Test is robust to low-quality SDCs, we study the effect of injecting 1000 random hashing SDCs candidates (in Section 5.1). Specifically, a random hashing SDC has a domain-evaluation function $f^d_{hash}(h_i, v) = h_i(v)$ where h_i is a hash function that randomly maps v to a real number between 0 and 1. Since hash functions do not correspond to any meaningful domain, these SDCs are inherently of low quality. We found that all adversarial SDC candidates are rejected by our statistical test, and consequently have no effect on our final results (e.g., with no false positive detections produced).

Additional results such as sensitivity to FPR budget, size budget, Cohen's h, and Wilson, can be found in [5] in the interest of space.

6.6 Quality on data-cleaning benchmarks

In addition to testing using large-scale real benchmarks collected in the wild (ST-Bench and RT-Bench), we also test SDCs learned using Auto-Test against 9 existing data-cleaning datasets used in prior work [13, 19, 28, 39, 46, 61], listed in Table 6. Our goal of this experiment is to test whether our learned SDCs can identify new constraints to complement existing constraints in these datasets, thereby augmenting existing data cleaning algorithms.

Table 7. Details of real SDC that are automatically applied on existing data-cleaning benchmarks, using Auto-Test. Many of these SDC constraints here offer new mechanisms to identify data errors, that are not possible using existing constraints from these benchmark data (e.g., note that existing constraints from these benchmark data do not cover many columns marked as "-", or existing constraints identify errors using complementary mechanisms, such as "2 letters" for `state` columns, while our SDC use ML-based CTA "state-classifiers" from Sherlock and Doduo, which are more fine-grained in detecting subtle errors).

Dataset	Column	Example values	Existing constraints in benchmark data	Auto-Test: New SDC constraints (pre-condition)	Auto-Test: New SDC constraints (post-condition)	New errors SDC can identify if present (not by existing constraints)
adult	race	white, black, others, ...	-	80% column values have their *Glove* distances to "red" < 5.5	values whose *Glove* distances to "red" > 7.5	(Typo): wite, blaack, ... (Incompatible): seattle, male, ...
adult	sex	female, male	-	80% column values have their *Glove* distances to "male" < 7.0	values whose *Glove* distances to "male" > 9.5	(Typo): femele, malle, ... (Incompatible): masculina, finnish, ...
beers	city	san francisco, columbus, louisville, ...	brewery id → city	80% column values have their *Glove* distances to "hawaii" < 6.0	values whose *Glove* distances to "hawaii" > 11.0	(Typo): louisvilla, seettle, ... (Incompatible): maine, 9th ave., ...
beers	state	or, in, ca, fl, ...	brewery id → state, state (2 letters)	80% column values have their *Sherlock state-classifier* scores > 0.5	values whose *Sherlock state-classifier* scores ≤ 0	(Typo): ax, xk, ... (Incompatible): us, xl, ...
food	facility type	restaurant, school, grocery store, ...	-	80% column values have their *Doduo type-classifier* scores > 4	values whose *Doduo type-classifier* scores < -1	(Typo): childern's service, koisk, ... (Incompatible): asia, dummy_type, ...
food	city	chicago, schaumburg, lake zurich, ...	-	80% column values have their *Glove* distances to "berlin" < 5.5	values whose *Glove* distances to "berlin" > 8.0	(Typo): chiago, buffolo, ... (Incompatible): upenn, mcdonald, ...
food	state	il, ilxa	city → state	80% column values have their *Doduo state-classifier* scores > 4	values whose *Doduo state-classifier* scores < -2	(Typo): xx, nt, ... (Incompatible): usa, tottenham, ...
hospital	sample	0 patients, 107 patients, 5 patients, ...	-	93% column values match pattern "\d+ \[a-zA-Z]+"	values not matching pattern "\d+ \[a-zA-Z]+"	(Typo): x patients, 3x patients,... (Incompatible): empty, sample_size, ...
hospital	state	al, ak	zip → state, county → state, state (2 letters)	80% column values have their *Sherlock state-classifier* scores > 0.5	values whose *Sherlock state-classifier* scores ≤ 0	(Typo): ax, xk, ... (Incompatible): us, xl, ...
hospital	hospital type	acute care hospitals	condition, measure name → hospital type	80% column values have their *Doduo category-classifier* scores > 4.5	values whose *Doduo category-classifier* scores < -1.5	(Typo): acute caer, clinix, ... (Incompatible): london, co. kildare, ...
hospital	emergency service	yes, no	zip → emergency service	80% column values have their *Glove* distances to "no" < 5.5	values whose *Glove* distances to "no" > 7.0	(Typo): yxs, nao, ... (Incompatible): emergency, 95503, ...
movie	id	tt0054215, tt0088993, tt0032484, ...	-	85% column values match pattern "\[a-zA-Z]+\d+"	values not matching pattern "\[a-zA-Z]+\d+"	(Incompatible): iron_man_3, dark_tide, ...
movie	duration	109 min, 96 min, 120 min, ...	-	93% column values match pattern "\d+ \[a-zA-Z]+"	values not matching pattern "\d+ \[a-zA-Z]+"	(Incompatible): 2 hr 30 min, nan, ...
rayyan	article created_at	[1/1/71, 4/2/15, 12/1/06, ...]	-	90% column values return true on function *validate_date()*	values that return false on function *validate_date()*	(Incompatible): nan, june, ...
soccer	position	defender, midfield, goalkeeper, ...	-	80% column values have their *Sherlock position-classifier* scores > 0.1	values whose *Sherlock position-classifier* scores ≤ 0	(Typo): strikor, forwrad, ... (Incompatible): difensore, goleiro, ...
soccer	city	cardiff, dortmund, munich, ...	-	80% column values have their *Sentence-BERT* distances to "panama" < 1.2	values whose *Sentence-BERT* distances to "panama" > 1.375	(Typo): cardif, munihei, ... (Incompatible): fl, 744-9007, ...
tax	state	ma, nv, ar, ...	zip → state, area code → state, state (2 letters)	80% column values have their *Sherlock state-classifier* scores > 0.5	values whose *Sherlock state-classifier* scores ≤ 0	(Typo): ax, xk, ... (Incompatible): us, xl, ...

Table 6 reports our results in terms of (1) column-level coverage, or new constraints that we discover using SDCs not in existing ground-truth, (2) column-level precision, or the fraction of new SDCs constraints judged as correct, (3) cell-level true-positives, or the number of erroneous cells that the new SDCs can detect, and (4) cell-level precision, or the fraction of erroneous cells detected by SDCs that are correct.

Column-level results. We can see that our approach can indeed discover new SDC constraints on 16 columns (not known in existing benchmark ground-truth), from a total of 81 columns, in which 94% new constraints are correct.

Table 7 lists all new SDCs found automatically on existing Data-cleaning benchmarks. Observe that some of these columns do not originally have applicable constraints in benchmark ground-truth (marked by "-"), in which case our SDCs auto-applied on such columns would clearly provide value, by enabling new mechanisms for error detection. While for the rest of the columns existing

Table 8. Example new errors detected by SDCs, marked in underline, that are not known or labeled as errors in existing benchmark ground-truth. For example, in the "hospital" dataset, a cell with value "empty" in the "sample" column (with typical values like "0 patients", "107 patients") are not marked in ground-truth; in the "food" dataset, a cell with misspelled "childern" is not marked in the ground-truth, etc. This shows the promise that SDCs can complement existing constraint-based cleaning to identify additional errors.

Dataset	Column	Example column values	Existing constraints in benchmark ground-truth	New SDC constraint (pre-condition)	New SDC constraint (post-condition)	New errors detected by SDC (not known in ground-truth)
hospital	sample	[0 patients, 107 patients, 5 patients, ...]	-	93% column values match pattern "\d+ \[a-zA-Z]+"	values not matching pattern "\d+ \[a-zA-Z]+"	"<u>empty</u>"
food	facility type	[restaurant, grocery store, catering, ...]	-	80% column values have their *Doduo type-classifier* scores > 4	values whose *Doduo type-classifier* scores < -1	"<u>childern's</u> service facility"
rayyan	article created_at	[1/1/71, 4/2/15, 12/1/06, ...]	-	90% column values return true on function *validate_date()*	values that return false on function *validate_date()*	"<u>nan</u>"

constraints do exist, our SDC can nevertheless still augment them. For example, on column "city" in dataset "beers", although there is an FD constraint between "brewery id" and "city", there are still many errors that cannot be reliably detected by the FD alone – e.g., the FD constraint cannot find errors for rows with a unique brewery id in the table, while SDC can help to detect errors such as typos (e.g., "seatle") and incompatible values (e.g., "9th ave") in such cases.

Cell-level results. At the cell-level, we can see from Table 6 that these automatically-installed new SDCs alone (without using any other constraints in benchmark ground-truth, which typically require human experts to program), can already identify 183 data values as errors, with an overall precision of 95%, when evaluated against the ground-truth clean data.

Interestingly, we would like to highlight that using these new SDCs enable us to uncover <u>*new errors not known or labelled in existing ground-truth*</u>, some of the example errors, and their corresponding SDCs, are shown in Table 8. Note that no constraints are programmed on these example columns in the existing benchmarks (indicated by "-" in the table), but the SDCs we automatically apply, can find typos (misspelled "childern"), and incompatibility (strings like "empty" and "nan" mixed in data columns) that are not known in existing ground-truth. We believe this demonstrates that SDC has the potential to augment existing data-cleaning methods, to identify new and complementary data errors not covered by existing constraints.

We want to stress that in expert-driven data-cleaning scenarios and with experts in the loop, existing data-cleaning formalism such as denial constraints are still way more expressive, such that *SDCs are not meant to outperform or replace existing methods in such settings* – this particular experiment is only meant to show that SDCs can serves as a new class of constraints (auto-applied to relevant table columns), that may complement existing data cleaning methods.

7 Conclusions and Future Work

In this work, we propose a new class of data-quality constraints that we argue are overlooked in the literature. We show that such constraints can unify diverse column-type detection methods in the same framework, and once learned from large table corpora using Auto-Test, can reliably apply to new and unseen tables.

Future directions include integrating SDCs with existing integrity constraints, and study how best to leverage them in the expert-driven data cleaning scenarios. Testing the coverage of our proposed method on specialized domains and corpora, is another direction of future work.

Acknowledgement

We thank three anonymous reviewers for their constructive feedback that helps to improve this paper. Qixu Chen and Raymond Chi-Wing Wong are supported by fund WEB24EG01-H.

References

[1] [n. d.]. Dataprep.clean: Curated validation functions for common semantic types. https://docs.dataprep.ai/user_guide/clean/introduction.html#userguide-clean.

[2] [n. d.]. Example UPC validation function (Python). https://en.wikipedia.org/wiki/Luhn_algorithm.

[3] [n. d.]. Excel clean data: an example demo. https://drive.google.com/file/d/1kIVLVOZQfZn2Dqd2M-fblo7EwpP_O4tw/view?usp=drive_link.

[4] [n. d.]. Excel: clean data with copilot. https://support.microsoft.com/en-us/office/clean-data-in-excel-7fe20d89-3f57-46d3-b659-e8f3ee853bda?ns=XLWAENDUSER&version=16.

[5] [n. d.]. Full technical report version of our paper, as well as code and data. https://github.com/qixuchen/AutoTest.

[6] [n. d.]. Google Sheet: Smart Cleanup feature. https://workspace.google.com/blog/product-announcements/connected-sheets-is-generally-available.

[7] [n. d.]. Google sheets. https://workspace.google.com/products/sheets/.

[8] [n. d.]. Microsoft Excel. https://www.microsoft.com/en-us/microsoft-365/excel.

[9] [n. d.]. TabLib: 627M tables and 867B tokens of context for training Large Data Models. https://www.approximatelabs.com/blog/tablib.

[10] 2024. validators - Python Data Validation for Humans. https://github.com/python-validators/validators/.

[11] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. 1999. *Modern information retrieval*. Vol. 463. ACM press New York.

[12] Laure Berti-Équille, Hazar Harmouch, Felix Naumann, Noël Novelli, and Saravanan Thirumuruganathan. 2018. Discovery of genuine functional dependencies from relational data with missing values. *Proceedings of the VLDB Endowment* 11, 8 (2018), 880–892.

[13] George Beskales, Ihab F Ilyas, Lukasz Golab, and Artur Galiullin. 2013. On the relative trust between inconsistent data and inaccurate constraints. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 541–552.

[14] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer.

[15] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 93–104.

[16] Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[17] Kaushik Chakrabarti, Surajit Chaudhuri, Zhimin Chen, Kris Ganjam, Yeye He, and W Redmond. 2016. Data services leveraging Bing's data assets. *IEEE Data Eng. Bull.* 39, 3 (2016), 15–28.

[18] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 1–58.

[19] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 458–469.

[20] Xu Chu, Ihab F Ilyas, Paolo Papotti, and Yin Ye. 2014. RuleMiner: Data quality rules discovery. In *2014 IEEE 30th International Conference on Data Engineering*. IEEE, 1222–1225.

[21] Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 1247–1261.

[22] Jacob Cohen. 2013. *Statistical power analysis for the behavioral sciences*. Routledge.

[23] Jacob Cohen. 2016. A power primer. (2016).

[24] Rémi Domingues, Maurizio Filippone, Pietro Michiardi, and Jihane Zouaoui. 2018. A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern recognition* 74 (2018), 406–421.

[25] Brian S Everitt. 1992. *The analysis of contingency tables*. CRC Press.

[26] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. 2010. Discovering conditional functional dependencies. *IEEE Transactions on Knowledge and Data Engineering* 23, 5 (2010), 683–698.

[27] Benjamin Feuer, Yurong Liu, Chinmay Hegde, and Juliana Freire. 2023. ArcheType: A Novel Framework for Open-Source Column Type Annotation using Large Language Models. *arXiv preprint arXiv:2310.18208* (2023).

[28] Congcong Ge, Yunjun Gao, Xiaoye Miao, Bin Yao, and Haobo Wang. 2020. A hybrid data cleaning framework using markov logic networks. *IEEE Transactions on Knowledge and Data Engineering* 34, 5 (2020), 2048–2062.

[29] Yeye He, Jie Song, Yue Wang, Surajit Chaudhuri, Vishal Anil, Blake Lassiter, Yaron Goland, and Gaurav Malhotra. 2021. Auto-Tag: Tagging-Data-By-Example in Data Lakes. *arXiv preprint arXiv:2112.06049* (2021).

[30] Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. 2019. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 International Conference on Management of Data*. 829–846.

[31] Joseph M Hellerstein. 2013. Quantitative data cleaning for large databases. (2013).

[32] Zhipeng Huang and Yeye He. 2018. Auto-detect: Data-driven error detection in tables. In *Proceedings of the 2018 International Conference on Management of Data*. 1377–1392.

[33] Yka Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal* 42, 2 (1999), 100–111.

[34] Madelon Hulsebos, Paul Groth, and Çağatay Demiralp. 2023. AdaTyper: Adaptive Semantic Column Type Detection. *arXiv preprint arXiv:2311.13806* (2023).

[35] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. 2019. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1500–1508.

[36] Andrew Ilyas, Joana MF da Trindade, Raul Castro Fernandez, and Samuel Madden. 2018. Extracting syntactical patterns from databases. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 41–52.

[37] Maria Kateri. 2014. Contingency table analysis. *Statistics for Industry and Technology* 525 (2014).

[38] Ken Kelley and Kristopher J Preacher. 2012. On effect size. *Psychological methods* 17, 2 (2012), 137.

[39] Zuhair Khayyat, Ihab F Ilyas, Alekh Jindal, Samuel Madden, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. 2015. Bigdansing: A system for big data cleansing. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 1215–1230.

[40] JooSeuk Kim and Clayton D Scott. 2012. Robust kernel density estimation. *The Journal of Machine Learning Research* 13, 1 (2012), 2529–2565.

[41] Edwin M Knox and Raymond T Ng. 1998. Algorithms for mining distancebased outliers in large datasets. In *Proceedings of the international conference on very large data bases*. Citeseer, 392–403.

[42] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2023. Table-gpt: Table-tuned gpt for diverse table tasks. *arXiv preprint arXiv:2310.09263* (2023).

[43] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. 2010. Annotating and searching web tables using entities, types and relationships. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 1338–1347.

[44] Yiming Lin, Yeye He, and Surajit Chaudhuri. 2023. Auto-bi: Automatically build bi-models leveraging local join prediction and global schema graph. *arXiv preprint arXiv:2306.12515* (2023).

[45] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *2008 eighth ieee international conference on data mining*. IEEE, 413–422.

[46] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective error correction via a unified context representation and transfer learning. *Proceedings of the VLDB Endowment (PVLDB)* 13, 11 (2020), 1948–1961.

[47] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A configuration-free error detection system. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. ACM, 865–882.

[48] Markos Markou and Sameer Singh. 2003. Novelty detection: a review—part 1: statistical approaches. *Signal processing* 83, 12 (2003), 2481–2497.

[49] Markos Markou and Sameer Singh. 2003. Novelty detection: a review—part 2:: neural network based approaches. *Signal processing* 83, 12 (2003), 2499–2521.

[50] Zhengjie Miao and Jin Wang. 2023. Watchog: A Light-weight Contrastive Learning based Framework for Column Annotation. *Proceedings of the ACM on Management of Data* 1, 4 (2023), 1–24.

[51] Tomas Mikolov. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[52] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. 1003–1011.

[53] Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. 2022. Can foundation models wrangle your data? *arXiv preprint arXiv:2205.09911* (2022).

[54] Felix Naumann. 2014. Data profiling revisited. *ACM SIGMOD Record* 42, 4 (2014), 40–49.

[55] Wei Ni, Xiaoye Miao, Xiangyu Zhao, Yangyang Wu, and Jianwei Yin. 2023. Automatic data repair: Are we ready to deploy? *arXiv preprint arXiv:2310.00711* (2023).

[56] Jinglin Peng, Weiyuan Wu, Brandon Lockhart, Song Bian, Jing Nathan Yan, Linghao Xu, Zhixuan Chi, Jeffrey M Rzeszotarski, and Jiannan Wang. 2021. Dataprep. eda: Task-centric exploratory data analysis for statistical modeling in python. In *Proceedings of the 2021 International Conference on Management of Data*. 2271–2280.

[57] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[58] Robin L Plackett. 1983. Karl Pearson and the chi-squared test. *International statistical review/revue internationale de statistique* (1983), 59–72.

[59] Prabhakar Raghavan and Clark D Tompson. 1987. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica* 7, 4 (1987), 365–374.

[60] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[61] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820* (2017).

[62] SciPy. 2024. SciPy optimize. https://docs.scipy.org/doc/scipy/reference/optimize.html.

[63] Jie Song and Yeye He. 2021. Auto-validate: Unsupervised data validation using data-domain patterns inferred from data lakes. In *Proceedings of the 2021 International Conference on Management of Data*. 1678–1691.

[64] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating columns with pre-trained language models. In *Proceedings of the 2022 International Conference on Management of Data*. 1493–1503.

[65] Yushi Sun, Hao Xin, and Lei Chen. 2023. Reca: Related tables enhanced column semantic type annotation framework. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1319–1331.

[66] Mihai Surdeanu, David McClosky, Julie Tibshirani, John Bauer, Angel X Chang, Valentin I Spitkovsky, and Christopher D Manning. 2010. A Simple Distant Supervision Approach for the TAC-KBP Slot Filling Task.. In *TAC*.

[67] David MJ Tax and Robert PW Duin. 2004. Support vector data description. *Machine learning* 54 (2004), 45–66.

[68] Michael E Tipping and Christopher M Bishop. 1999. Probabilistic principal component analysis. *Journal of the Royal Statistical Society Series B: Statistical Methodology* 61, 3 (1999), 611–622.

[69] Pei Wang and Yeye He. 2019. Uni-detect: A unified approach to automated error detection in tables. In *Proceedings of the 2019 International Conference on Management of Data*. 811–828.

[70] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[71] Edwin B Wilson. 1927. Probable inference, the law of succession, and statistical inference. *J. Amer. Statist. Assoc.* 22, 158 (1927), 209–212.

[72] Catharine Wyss, Chris Giannella, and Edward Robertson. 2001. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances extended abstract. In *Data Warehousing and Knowledge Discovery: Third International Conference, DaWaK 2001 Munich, Germany, September 5–7, 2001 Proceedings 3*. Springer, 101–110.

[73] Junjie Xing, Yeye He, Mengyu Zhou, Haoyu Dong, Shi Han, Dongmei Zhang, and Surajit Chaudhuri. 2024. Table-LLM-Specialist: Language Model Specialists for Tables using Iterative Generator-Validator Fine-tuning. *arXiv preprint arXiv:2410.12164* (2024).

[74] Mohamed Yakout, Laure Berti-Équille, and Ahmed K Elmagarmid. 2013. Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 553–564.

[75] Cong Yan and Yeye He. 2018. Synthesizing type-detection logic for rich semantic data types using open-source code. In *Proceedings of the 2018 International Conference on Management of Data*. 35–50.

[76] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çağatay Demiralp, and Wang-Chiew Tan. 2019. Sato: Contextual semantic type detection in tables. *arXiv preprint arXiv:1911.06311* (2019).

[77] Chen Zhao and Yeye He. 2019. Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *The World Wide Web Conference*. 2413–2424.

Received October 2024; revised January 2025; accepted February 2025