Is this Real? Generating Synthetic Data that Looks Real

Miro Mannino, Azza Abouzied
New York University Abu Dhabi, UAE
{miro.mannino, azza}@nyu.edu

ABSTRACT

Synner is a tool that helps users generate real-looking synthetic data by visually and declaratively specifying the properties of the dataset such as each field's statistical distribution, its domain, and its relationship to other fields. It provides instant feedback on every user interaction by updating multiple visualizations of the generated dataset and even suggests data generation specifications from a few user examples and interactions. Synner visually communicates the inherent randomness of statistical data generation. Our evaluation of Synner demonstrates its effectiveness at generating realistic data when compared with Mockaroo, a popular data generation tool, and with hired developers who coded data generation scripts for a fee.

CCS Concepts

•**Human-centered computing** → **Interactive systems and tools;**

Author Keywords

Data Generation; Mixed-Initiative UI; Uncertainty Visualization

INTRODUCTION

Data generation is an important tool for a variety of users from teachers who teach statistical methods or data science, to software designers and developers who demo their systems. Often these users resort to synthetic data (i) when access to real data sets is difficult: medical or school records, income data, etc. are understandably private, (ii) when real data sets are not of the right granularity: publicly available data through initiatives such OpenData.gov are often aggregated, (iii) when real data sets are not of the right scale: data owners often provide restricted access to a small subset of their larger and richer first-party data, or (iv) when real data sets are incomplete: a poor data sample may be missing key illustrative trends or key fields entirely. We first motivate our work on realistic data generation for specific use cases and we then explain the technical challenges of synthesizing real-looking data.

UIST '19, October 20-23, 2019, New Orleans, LA, USA

 ISBN 978-1-4503-6816-2/19/10…$15.00
DOI: https://doi.org/10.1145/3332165.3347866

Motivating Use Cases

In pedagogy, "integrating real data with a context and purpose" is one of the six recommendations put forward by the American Statistical Association (ASA) on how to best teach introductory statistics courses in college [8]. By posing and answering questions on real data sets, ASA hopes to improve the research and problem-solving mindset of students. However, as Kim et al., who are experienced data science educators, note the "two goals, to 'minimize prerequisites to research' while also using real-world data are in direct conflict with each other. On the one hand, one cannot expect novices to immediately tackle many raw datasets as they exist 'in the wild,' thereby placing barriers to research at a time where there is high risk of alienating them. On the other hand, to present students with datasets that are overly curated would betray the true nature of the work done by statisticians and data scientists ... A careful and thoughtful balance between these two goals is essential when preparing datasets for use in introductory statistics and data science courses" [21].

Kim et al. argue that teaching data must instead satisfy three R's: "be *rich* enough to answer meaningful questions with, be *real* enough to ensure that there is context and be *realistic* enough to convey to students that data as it exists 'in the wild' often needs pre-processing" [21]. They provide the well-curated fivethirtyeight R real datasets to satisfy these three requirements; the *tame* datasets, adapted from FiveThirtyEight's journalistic data [10], allow students to focus on key learning concepts as they gradually increase the realism of the data sets. A major shortcoming of such data sets, tame or otherwise, is their lack of context or cultural appropriateness: ASA's example data sets include Iowa's real-estate prices, diamond pricing data, US teacher salaries and SAT scores; Fivethirtyeight's datasets are overwhelmingly US-centric with data on the NBA, the NFL, March Madness, and US-politics [10]. Thus, while real, these datasets lack context outside the US or the western world. Many countries do not have the same data-driven initiatives as those in the US leading to an absence of alternative data sets. Simply relabelling a US data set, such as US teacher salaries, for another country does not work due its unique state-specific nature, governing socio-political factors, educational system, etc. Moreover, many tame datasets often do not allow the demonstration of complex statistical methods: stats.stackexchange is awash with synthetic data examples to either illustrate a data problem or a solution. *Data generation can allow educators to create data sets that not only satisfy the three R's but are also appropriately contextualized for the target student group and illustrative of deeper statistical concepts.*

In technology, one can draw parallels between demoing a software-product to potential investors or customers and staging a house to sell it [27]. In staging a software-product, software designers often need to populate their systems with data that their customers can relate to instead of the unimaginative 'test,' '1234,' and 'Lorem Ipsum ...' values for string, numeric and text fields respectively. This is even more critical for data analytics or decision-making software, where it pays off to emphasize the data exploration, visualization, prediction features, etc. using rich and realistic data. *By synthesizing data instead of using real data, one can minimize distractions stemming from dirty data or unanticipated trends, and emphasize software features by ensuring that the demo data set is rich and complete with respect to the features demoed.*

Assumptions about Target Users

In addition to the users from our motivating use cases (data science or statistics instructors, software developers and demonstrators), we anticipate a much larger user base that would benefit from expressive, and easy realistic data generation. We make the following assumptions about our target user base:

A1 *Our users view realistic data generation as secondary but essential to a primary task.* A teacher's primary goal may be to expose students to data-driven decision-making. Thus, she desires a quick, easy to learn, and effective tool for data generation that can help her focus on her primary goal of building an overall lesson plan.

A2 *Our users have a wide range of programming experience* from novices to experts.

A3 *Our users have a wide range of statistical literacy.* Some users prefer to specify precise statistical models to sample data from; others prefer to visually specify or validate the shape of the generated data and its relationships.

A4 *Our users have a conceptual model of the data that evolves during specification.* Users will often start with initial data properties and relationships and will iteratively refine these as they visually preview the generated data.

The Technical Challenges of Generating 'Real' Data with Existing Methods

Rich, real, realistic data generation is a technically complex task. As we empirically demonstrate, experienced developers can spend hours writing a data generation script for a relatively small data set, on the order of ten fields, and still produce data sets that are not real enough. Software tools like Mockaroo[1] drastically simplify the process but are not sufficiently expressive. Given the assumptions of our target users, data generation is complex because we need to:

1. Correctly implement the data-generating process, or the statistical model, of real data. Randomness is an inherent property of many real datasets. We introduce noise and errors during the data collection and measurement process. Natural and complex systems are usually open and are difficult to model completely and precisely, hence we rely on stochastic processes and random distributions to best describe them. Thus, statistical modeling goes hand-in-hand with realistic data generation. Generating data, however, from well-known, or custom, statistical distributions is not straightforward. Users have to correctly parameterize existing generators or look-up values from the cumulative distribution function of the intended distribution using uniformly distributed random values (See A2, A4). Finally, users often need to recognize that what they specify may not be exactly what they get. A frequency histogram constructed over a small sample of normally distributed data, for example, will not have the perfect bell shape commonly associated with normal distributions (See A3).

2. Correctly implement the many (probabilistic) relationships between fields. Dependencies across fields can be due to domain dependencies, such as when co-generating cities and countries, mathematical expressions, such as `age = year(today) - year(dateofbirth)`, and statistical relationships between random variables, such as men on average being taller than women. Dependencies introduce ordering constraints on how we generate data, which if not implemented correctly can make generation computationally intractable or result in unreal data (See A1, A2).

3. Carefully introduce errors and missing values and ***easily scale[2] up or down the size of the data set while preserving its properties***. Corrupting a certain proportion of values within a field often has cascading effects on other fields, which could break the data generation pipeline. Scaling down a generated dataset may result in the biased sampling of domains: in our evaluation, one developer sequentially selected names from a list of roughly 8000 lexicographically ordered names up to the data size, which produced small datasets with only names starting with 'A.' Scaling up a generated dataset may result in unintended repetitions, poor performance, etc. (See A1)

Synner's Design Principles

To address the above challenges, we designed Synner with the following design principles:

1. Visual Lifting & Declarative Specification. Heer et al. coin the term visual lifting to describe the interface design principle of *lifting* a domain specific language into an isomorphic higher-level visual language that is more natural for users [13]. Synner lifts from scripts in a data generation language to a more intuitive visual domain: Synner generates, samples and presents data at every user interaction in a familiar spreadsheet visualization, along with histograms and basic statistics such as mean and standard deviation. These visualizations enable users to quickly assess the quality and progress of their data generation process. The interface cues users on how to add fields, dependencies and even what distributions to use or which domains to pick data from. Specification is not only visual — users can even draw custom univariate distributions or bivariate relationships — but also declarative. Users only focus on what the data looks like but not how it is generated: they do not specify the order of generating data

[1] In the related work section, we describe how users specify data generation tasks with Mockaroo and we describe its limitations.

[2] In this work, we examine data scales of the order of hundreds of records to hundreds of thousands of records (See Conclusion for more details).

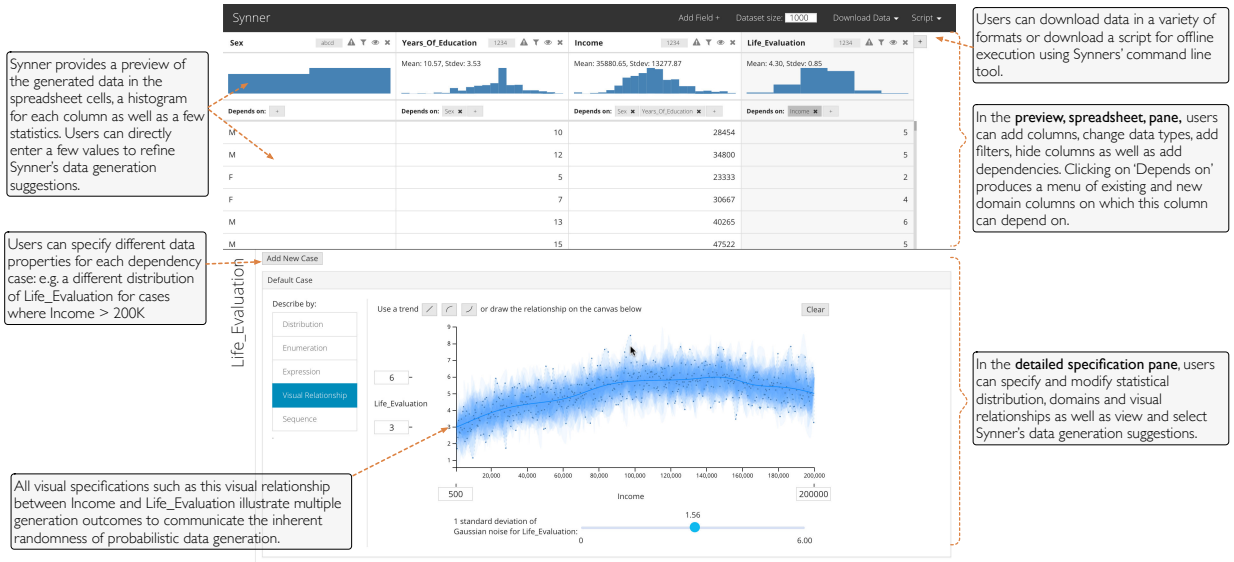

Figure 1. Synner's user interface.

fields, or when to optimally apply selection operations (e.g. when wishing to select only a subset of the generated data), or how to sample from a statistical distribution, or how to jointly generate values from related domains such as city and country. Visually lifting from data generation scripts narrows the gulf of evaluation as users can immediately evaluate how real their data looks as they build their dataset. Visual and declarative specification narrows the gulf of execution as users can focus on the properties and relationships of their data and not how to create executable and scalable generation scripts.

2. Example-driven Interaction. To further narrow the gulf of execution, Synner infers what users wish to generate from column labels or values manually inserted into the data spreadsheet. Synner suggests an ordered list of the top five generator recommendations (e.g. it suggests the domain 'city' from entries like 'Milan', 'Cairo' in the spreadsheet or 'Location' as the column label; It suggests a normal distribution with mean μ and standard deviation σ for a few numeric entries, X, entered into the spreadsheet where $\mu = mean(X)$ and $\sigma = stdev(X)$). It also suggests dependencies or relationships between domains, e.g. 'city' depends on 'country'. Synner's example-driven interaction helps users who are still unfamiliar with Synner's features, or who may wish to explore what data generation options are available and what domains exist in Synner's database, to quickly and easily specify their dataset properties.

3. Communicating Randomness. Designing data visualizations that omit the inherent uncertainty in real data and create the illusion of preciseness can often mislead users and lead to poor decisions [4, 19, 17]. Similarly, designing a visual data generation specification tool that does not communicate the uncertain nature of probabilistic data generation can mislead users and decrease trust in the tool. This mistrust can be exacerbated by our preconceived misconceptions of randomness. For example, in a publicized classroom exercise at UC Berkeley, college students at a statistics class, would regularly alternate between heads and tails with very short consecutive runs when faking data for 100 coin tosses; data from 100 actual coin tosses, however, contain longer consecutive runs. People mistakenly assume that a Bernoulli process self-regulates and that long consecutive runs are extremely unlikely [3]. Thus, communicating a range of plausible probabilistic data generations for a given specification is important to not only improve user trust when the outcomes differ from user expectations, but also to allow users to fine-tune their specifications.

4. Separation of Concerns. Unlike outliers, errors such as out-of-bound values, encoding errors, or missing data are not intrinsic to the data-generation process. With this point of view, we separate the introduction of realism in the form of errors and missing data from the generation specification. Errors and missing values can be introduced post-hoc as corruptions to the dataset that do not affect dependencies or data generation order, for example. This viewpoint limits the class of errors one can easily introduce with Synner, to only those produced by independent corruption processes. That said, users can still choose to directly specify the distribution of erroneous or missing values within the dataset, as if it were part of the data, with the help of **cases**, a feature we describe in the following section.

We describe how we abide by these design principles in Synner. Table 1 summarizes our design objectives & principles given our user assumptions, and contrasts them with existing methods, which are further discussed in related works. As we walk through Synner, we refer back to these enumerated objectives. In the supplementary material, we describe the technical systems aspects of Synner that support its interaction features.

We also demonstrate that Synner enables users to easily and quickly complete data generation tasks when compared with coding the task or using Mockaroo.

	Objectives & Principles	Assumptions	Synner	Mockaroo	Bayesian Modeling/Inference Tools (JAGS/BUGS)	DB Testing & Data Generation Languages
O1	Visually lift the generation specification into a higher-level visual language and interface that is more natural for users	A1 A2	yes	*partial*	*partial*	no
O2	Interactively and visually preview summaries of generated data to enable quick validations of realism	A1 A3 A4	yes	no	no	no
O3	Communicate randomness in the data generation process to enhance trust	A3	yes	no	no	no
O4	Declaratively specify properties of the data. Hide implementation and optimization details on how to sample from distributions, domains, or how to order generation operators for efficiency, etc.	A1 A2	yes	*partial*	yes	*some/partial*
O5	Suggest specifications for distributions, domains and relationships from examples and interactions through a mixed-initiative UI for effectiveness	A2 A4	yes	no	no	no
O6	Separate corruption processes for errors and missing values from data generation processes	A2	yes	no	no	no
O7	Infer model parameters from sample data and sample new simulations from models		*no*	*no*	*yes*	*some*

Table 1. Synner's design principles and objectives. We list whether (some of) these methods provide partial or no support for a given objective. The related word section provides further details.

It is important to point out that many big data benchmarking systems have often focused on generating big data that satisfy the three V's: volume, velocity and variety, ignoring realism [12]. In a recent survey of such systems, Han et al., emphasize that *veracity* is the fourth, albeit unconsidered, essential 'V' of big data benchmarking systems: "Veracity reflects whether the data used in big data systems conform to the inherent and important characteristics of raw data. This property is important to guarantee the reality and credibility of benchmarking results" [12]. Of the 16 data benchmarking systems studied, however, only 5 systems *partially consider* veracity by either using statistical distributions to generate the data or ensuring the raw data format (table, graph, text) is preserved [12]. *Han et al. explain that perhaps the absence of standard mechanisms to measure the degree of conformity of a synthetic data set to a real one is partly to blame for the lack of veracity in these systems [12].*

Thus, in addition to designing Synner to enable easy and expressive data generation, we put forward a user-centered method for evaluating how fake or real a data set is: a data set is real enough if it passes user-provided checks specific to a given dataset. We show that datasets generated by Synner outperform developer- or Mockaroo- generated data with respect to these checks.

We conclude this paper by discussing related works and their influence on Synner's design and discussing current limitations that we hope to address in the future.

GENERATING DATA WITH SYNNER

To ground our discussion of Synner's interface and design, we walk through how Imani, an Economics professor who teaches at a small college, generates a dataset to allow her first-year quantitative methods students to analyze the relationship between happiness and income. She bases her generated dataset on the results of a recent study that explores income and happiness across the world [18] and uses the Gallup World Poll dataset [1], which is not publicly available. She wishes to generate roughly 5000 data points. She hopes her students would analyze and visualize the dataset, which she explains represents a fictitious nearby nation, to explore and discuss issues of income- and education- inequality, as well as the relationship between income and happiness. She sketches the data schema in Figure 2.

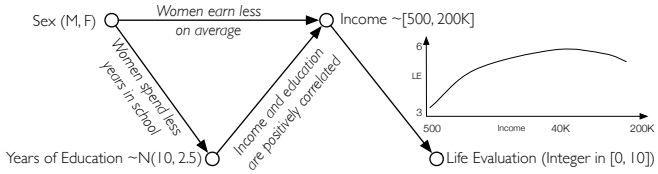

Figure 2. Imani sketches this schema, which describes the dataset she wishes to generate and the relationships between the different fields.

Synner's interface is split into main panes: a *preview, spreadsheet pane* at the top, which allows users to add columns, change data types, add dependencies, enter a few example entries to refine Synner's suggestions as well as filter rows, hide columns and add errors or missing values, and a *detailed specification pane* at the bottom that allows users to visually specify and modify statistical distributions, domains and visual relationships, as well as view and select Synner's suggestions for data generation (See O1, O4, O5). Figure 1 illustrates Synner's interface after Imani completes her data generation task.

Domains & Enumerations

Imani begins by creating the sex column: she selects the data type as 'string' and enters the column label: 'Sex'. Even with only these interactions, Synner suggests the domain Sex, which has a natural frequency of 50% male, 50% female. Synner's database has tables describing several domains including city, country, name, last name, plant, as well as relationship tables that relate a pair of domains such as city and country. Users can upload their own domain and relationship tables as well into the database.

Imani enters 'M' and 'F' in the cells of the Sex column and Synner suggests an enumeration of values 'M' and 'F'. Imani prefers this suggestion as it allows her to visually control the proportion of males and females in her dataset: she sets the ratio of male to female at 3:2. Synner interprets the specification as a discrete probability distribution on the enumeration (See Figure 3).

Figure 3. Imani specifies the ratio of males to females in her dataset with the help of an enumeration.

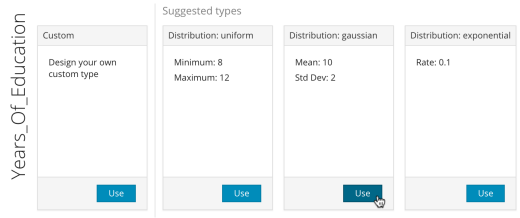

Figure 4. Synner suggests different distributions from a few examples entered by Imani in the Years of Education column

Synner can also generate strings from regular expressions. For example, if Imani specifies `M|Male` in the enumeration, Synner will randomly generate either 'M' or 'Male' strings. Synner converts the expression into an automata and then randomly walks through the automata to generate matching strings[3].

Example-driven interaction

Imani then creates the years of education column with an integer data type. She enters a few entries in the cells and Synner produces the suggestions in Figure 4.

Synner employs multiple algorithms to determine the best suggestions for a given column label, data type and column-entries (See O5). Synner first searches its domain database for partial matches between table names and column labels using an edit-based string similarity score. This allows Synner to suggest the domain 'name' and 'surname' when the user labels a column as 'First Name.' Synner then finds matches between example entries and all values within the domain tables using a k-nearest neighbor matching algorithm. If there are no matching domains in the database, Synner also suggests an enumeration with the entries provided by the user: the frequency of each entry is used to set the initial proportions of each value. Finally, for numeric or date/time data types, Synner suggests statistical distributions with parameters inferred from the few user-provided examples, or sequences such as serial numbers, serial dates, times, etc. with gaps that match the gaps between the user-provided entries.

Synner's suggestions extend beyond the properties of a single column: when a user clicks on the 'depends on' menu for a particular column, Synner also suggests related fields from the domain database. For example, if a user creates a column of country names, the 'depends on' menu will suggest other fields, which may have not yet been created by the user, from

[3]Synner supports all regular expression operators including $*$; it terminates a string generation if it executes for more than a pre-configured time limit.

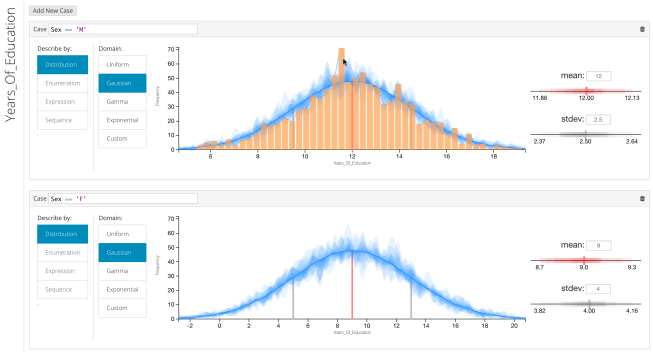

Figure 5. Imani specifies the relationship between years of education and sex using dependencies by cases. Synner illustrates the many plausible data generation outcomes using the Layers visualization. Hovering over a layer reveals the underlying histogram.

the domain database such as 'city', or 'continent'. There is no strict hierarchy for such dependencies, which allows 'country' to depend on 'continent' and vice-versa. To enable these rich relationships, the domain database stores each domain in an independent table and then stores pairwise relationships in a relationship table. To determine the available relationships for a domain, Synner computes the transitive closure on pairwise relationships. During data generation, Synner materializes relationships such as one between city and region by computing a join on the available intermediate pairwise relationships, such as those between city and country, and country and region, and then samples data from the join. See Figure 3 in the supplementary material for an example of these tables and the computed transitive closure. Through a simple csv upload interface, one can add more domains and relationships to Synner's database.

We find that our simple strategies for supporting example-driven interaction quite effective: as we show later in our evaluation of Synner, users often picked Synner's recommendations and tweaked them (e.g. changed a distribution's parameters) instead of manually specifying the data properties using Synner's menus.

Communicating Randomness

Imani selects the normal distribution from Synner's suggestions and adjusts the mean and standard deviation to 10 years and 2.5 respectively. Synner communicates to Imani the outcomes of many plausible data generations (See O3) using the Layers visualization in Figure 5.

To produce the *Layers* visualization, we generate 20 samples with as many data points as the dataset size, up to a configurable limit to ensure interactive performance, by sampling from the specified data distribution. We then construct a histogram with 50 equal-width bins for each sample. We connect the midpoints of each bin's frequency to form a smooth curve. We shade the area between the curve and the curve of an ideal histogram, one constructed from the distribution's cumulative density function, with a low opacity color. More opaque regions in the visualization represent overlapping areas. This visualization not only illustrates plausible histograms but also

illustrates which ones are more likely than others through opacity. On hovering over a layer, Synner overlays the actual histogram that created the layer. We believe this novel visualization is easier to interpret and more effective at communicating the outcomes of data generation than less-grounded visualizations that present abstract measures such as confidence intervals or probability density functions. We apply Layers to any visualization that presents statistical distributions including enumerations and visual relationships.

We conducted a mini user study with 32 participants, undergraduate students who completed an introductory statistics class, to determine the most effective visualization for communicating randomness. We compared Layers with three other alternatives presented in Figure 6:

1. *Min-Max bounds:* Like Layers, we generate 20 equal-width histograms for 20 samples. We shade the area between the histogram with the lowest frequency and the one with the highest frequency.

2. *Lines*: We only show the curves of the 20 sample histograms. In this visualization, the area surrounding the ideal visualization is tightly packed with lines. This visualization is influenced by Boukhelifa et al.'s use of *sketchiness* as a visual variable to communicate impreciseness in that surrounding the ideal histogram are multiple sketches of plausible generated histograms.

3. *Inferred lines*: This is perhaps the most abstract visualization we evaluated: for each of the 20 samples, we inferred the most-likely parameters of the distribution and then plotted the curve associated with the ideal histogram of the distribution with those parameters. In Figure 6, each line represents a slightly different Gaussian distribution than the one specified: each line has a different mean or standard deviation derived from one of the 20 samples.

In our study, we divided the participants into four groups to evaluate the effectiveness of each visualization in a between-subjects study. In each group, we explained to the participants how the visualizations are generated and how to interpret them. We then tested three different statistical distributions arising in three data generation scenarios: Gaussian for employee ages with a mean of 30 years and a standard deviation of 5, Exponential for time to sell laptops with a rate of 1 week, and Gamma for call center waiting times with shape = 3 and scale = 2 for a mean waiting time of 6 minutes. For each scenario we presented in randomized order four histograms of datasets that fit within the distribution to varying degrees: a good fit with high p-values for the Chi-squared goodness-of-fit statistic, adequate fit with a borderline p-value slightly higher than 0.05, a poor fit with a p-value slightly lower than 0.05 and one with p-value close to zero. Each participant was asked to decide, given the visualization of the specified distribution and the histogram, whether the dataset visualized by the histogram was generated by the specified distribution or not. Figure 6 presents the results of the experiment[4]. Since we wanted a visualization that allowed users to correctly discern the quality of a generated sample irrespective of the distribution, we chose *Layers*: users generally accepted good fits and rejected poor fits. We also considered Min-Max bounds but users were likely to reject good fit samples simply because a bar fell out of the shaded area.

Note that even though our participants understood that a finite-sample distribution can introduce randomness and the degree of this randomness depends on sample size[5], they still found our visualizations helpful in determining fit. By visually recognizing that a sampling distribution can have multiple outcomes, some of which are less than ideal, users are more likely to trust Synner's generated data even if it deviates from their conceptual model.

Dependencies by Cases

Imani now specifies that years of education depends on sex using the 'depends on' menu at the top of the column. This allows Imani to specify the following two cases in the bottom pane: if the sex is female, then years of education is normally distributed around 9 years with a standard deviation of 4, and if the sex is male then years of education is normally distributed around 11 years with a standard deviation of 2.5. She tries different parameters before she settles on these values (See Figure 5). She also examines the histogram above the years of education column to make sure that she is content with the overall distribution of values. Synner lets Imani specify dependencies declaratively (See O4). However, to ensure that Imani does not introduce cyclical dependencies, Synner displays an error message if Imani tries to make sex dependent on years of education either directly or indirectly as dependencies are transitive. Synner uses dependencies to determine the order of data generation by constructing a directed acyclic graph (DAG) from all dependencies.

Dependencies by Visual Relationships & Expressions

Imani now adds a new column for Income with an integer data type. She declares that Income depends on sex and years of education. She creates two cases for each sex and for each case she specifies the relationship between years of education and income visually as a linear relationship where the mean income is centered on the line (See O1). She then adds some Gaussian noise with a standard deviation of 13,000 dollars. She creates two such linear relationships: the female linear trend is shifted lower than the male one. Alternatively, Imani could have specified two mathematical expressions to describe the relationship between years of education and income for each sex.

Finally, she adds a new column for Life Evaluation — a 0-10 score on Cantril's Self-Anchoring Striving Scale [7], with 0 representing the 'worst possible life' and 10 representing the 'best possible life'. Using the plot of life evaluation against income by world regions, Imani declares that Life Evaluation depends on Income and sketches the following visual relationship between income and life evaluation. As she controls how much noise is added to the relationship, Synner communicates

[4]Given the relatively small size of only eight users for each visualization group, we interpret the results of Figure 6 qualitatively to guide the design of Synner, and we do not test for statistically significant differences across the groups.

[5]Sampling distributions were covered in the introductory statistics course.

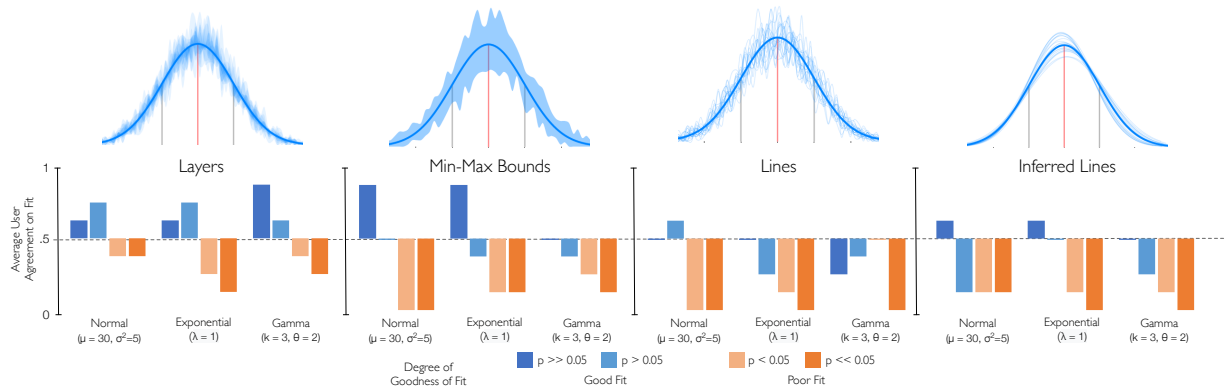

Figure 6. Results of a focused user study on how best to visualize the uncertainty associated with generating data for a given distribution.

several plausible data generation outcomes to her. On hovering at a layer, Synner reveals a scatterplot that shows the distribution of data points for that layer (See Figure 1).

Imani is content with her overall dataset but she notices that some of the values in Life Evaluation fall below 0 or are above 10. She decides to filter out these points using the selection operation. As she types her criteria, Synner also highlights the rows that remain after the operation (See Figure 7). Synner decides whether to push down selection operations or apply them after generating the dataset to ensure efficient runtime performance.

In the supplementary material, we further discuss the semantics of dependencies.

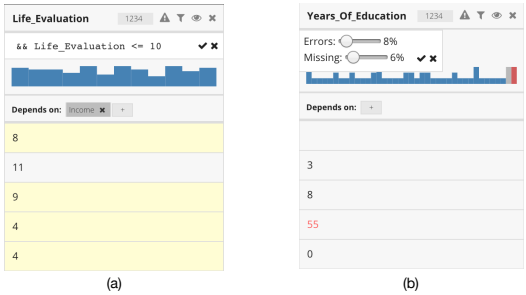

Figure 7. (a) Synner highlights rows that satisfy the selection operation. On accepting the selection criteria, non-highlighted rows will be filtered out, (b) Synner adds two bars to the overall column histogram to indicate the proportion of missing or error values.

Missing Values, Errors & Outliers

Imani can also add errors, or missing values to individual columns within her dataset by specifying the proportion of missing or erroneous values (See Figure 7). Synner only corrupts the values of a field after completely generating the data set to ensure that missing values and errors do not cascade to dependent fields or break the data generation process (See O6).

Imani can also add more complex errors beyond the default ones such as outliers, or broken dependencies: Imani introduces an indicator column and sets the proportion of its value being true to 0.1%. She specifies that Income now also depends on the indicator. She adds a case that when true, Income is described by an outlier distribution with a mean income of 500K, that is independent of sex or years of education.

Imani can either download her dataset in CSV or JSON format or save the declarative data generation script to upload later into Synner's interface to further modify it, or directly execute it using our command line tool to get a dataset of any size.

EVALUATION

To evaluate whether Synner's design supports realistic data generation, we conducted a within-subjects study comparing Synner with Mockaroo, and a between-subjects study with hired, experienced, developers. Our main hypotheses were:

H1 Users will spend less time specifying more complex data generation tasks in Synner than with Mockaroo or with custom scripts (even when recruiting experienced developers).

H2 Users will generate more realistic data sets with Synner than with alternatives as judged by an external group of realism *checkers*.

Our findings support both hypotheses. In the following sections we provide details on participants, experimental methods and analysis.

Participants and Methods

The End-Users. We conducted a within-subjects user study of Synner against Mockaroo with 18 participants from an academic institute: 15 undergraduates, 3 graduate students; 14 have a computer science background. Only 4 participants had some experience generating data with python, Matlab, or online statistical generators. We believe these participants are generally representative of users who may wish to use data generation tools. For example, of the 14 CS students, some may demo software tools in the future.

We trained the participants on each tool. We presented a video for each tool that demoed several features and briefly described the syntax of the expression language for both tools. We paused the demonstration regularly to respond to questions, to clarify any confusions and to make sure that the users can use the features themselves. We concluded each 10-15 minute

tutorial with a 15 minute play task, where we asked users to generate a toy dataset of buildings with several dependencies such as one between building height and year of construction and one between number of residential floors and height. They also added domain data such as city, or names to assign building names. We answered questions during this process and guided users on how to use certain features.

The Developers. We also recruited ten participants to develop data generation scripts (5 from Freelancer.com and 5 CS graduate students). We selected experienced Freelancer developers by ensuring they had a customer rating above 4.8/5. Also, we selected graduate students who had experience synthesizing data for experiments or demos. *All ten participants chose to write their scripts in Python*. These developers were financially compensated for their code. The average payment for Freelancer developers was 43 USD. All graduate students received 50 USD.

Data Generation Tasks. We asked all participants to generate three synthetic datasets:

1. ***Birth Records (BR)***, with fields: name, last name, date of birth, gender, mother's name, mother's last name, father's name, father's last name.
2. ***Train Schedules (TS)***, with fields: departure city, departure country, departure time, arrival city, arrival country, arrival time. We limited the countries to only those within the western European region (Austria, Belgium, France, Germany, Liechtenstein, Luxembourg, Monaco, Netherlands, Switzerland).
3. ***Age & Height Pediatric Records (AH)***, with fields: gender, age, and height. Age is between 2-16. We provided the CDC growth charts as a reference on how age and height are correlated.

We asked users to strive for realism. Other than the information above, we did not provide additional specifications or requirements. We randomized the order of tasks and, to alleviate learning effects, we randomized which tool participants generated the data with first. We also asked the participants to spend up to two minutes before using either tool to draft on paper the properties of the different fields and how they relate to each other. This minimized the time spent thinking about the dataset's properties while using each tool. We also observed that users tried to specify the properties they drafted independent of their failures or successes specifying them in the previous tool.

We provided developers the same minimal specifications, access to plain text lists of female and male names, cities by country for the nine western European countries, and growth charts. We also asked them to strive for realism. We required their scripts to be executable from command line with dataset-size passed as a command-line parameter. We required the scripts to output the datasets as comma-separated value files. We provided an example of how we intend to run the script in command line and a sample output csv file for reference.

Each task explored different data generation challenges. For BR, most participants tried to create (i) a simple uniform distribution for gender, (ii) dependencies to ensure gender-typical names, and (iii) relationships to reflect the cultural norm of sharing a last name with at least one of the parents. For TS, most participants tried to create (i) domain dependencies to ensure that cities exist within their respective countries, and (ii) statistical (or formulaic) relationships between departure time and arrival time. Finally for AH, most participants tried to create the complex statistical relationship between age and height as visually illustrated by the growth charts.

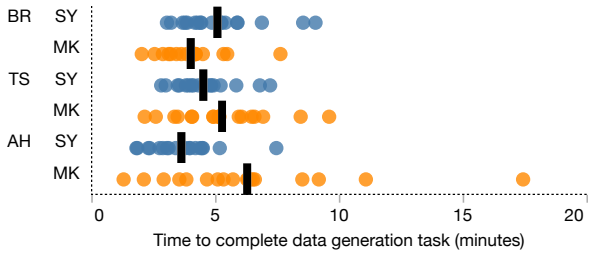

Figure 8. For each of the three tasks, we present each user's task completion time on Synner (SY) and Mockaroo (MK). Black bars show mean task completion times.

In terms of task complexity with respect to Mockaroo, BR was least complex as the domain dependency between names and gender is automatically inferred and applied without user intervention. TS was slightly more complex as it required the specification of mathematical expressions to relate departure and arrival times and AH was most complex.

We limited the evaluation to these specific tasks because they can be expressed in Mockaroo. Even slight modifications to the tasks made them too complex to immediately express in Mockaroo. For example, it is not possible to restrict departure city to one set of countries, and arrival city to a different set of countries: this currently breaks domain dependencies. Specifying gender using custom lists with a non-uniform probability distribution makes it impossible to assign gender-typical names. To recreate the simple case dependency, of years of education on gender, in Imani's dataset with Mockaroo, one would have to create two hidden columns with different normal distributions for each gender, with values pulled from one column, $\sim \mathcal{N}(9,4)$, for 'females' and the other column, $\sim \mathcal{N}(10,2.5)$, for 'males': this convoluted trick took roughly an hour for us to figure out! Finally, expressions in Mockaroo are written in Ruby: most of our participants were not familiar with Ruby's syntax. In Synner, expressions are written in JavaScript and an auto-suggest feature allows users, who are not familiar with JavaScript to easily specify them. We also did not evaluate adding errors or missing values as we found Mockaroo's behavior inconsistent for missing values: a missing value may or may not affect a dependent value.

Comparative User-Study Results

Task Completion Time

Figures 8 and 9 illustrate each participant's task completion time for each task and tool or developer group. Writing data generation code is two orders of magnitude slower than using GUI tools to generate data.

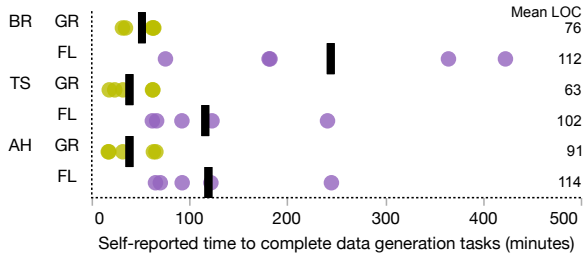

Figure 9. For each of the three tasks, we present each developer's task completion time separated by group, Freelancer developers (FL) and graduate students (GR). Black bars show mean task completion times. We show the mean lines of code (LOC) for each task as well.

To evaluate completion times for tool users, we performed a two-way repeated-measures ANOVA with task and tool-used as independent factors. We log-transformed the completion times to better approximate a normal distribution[6]. We found a significant interaction effect $F_{2,34} = 8.153, p = .001$. We find the following simple main effects: *for the least complex task (BR) and the most complex task (AH), there was a statistically significant difference in completion times with* $F_{1,17} = 6.537, p = .02$ *and* $F_{1,17} = 7.939, p = .01$ *respectively.* We observed that users often hit a 'cliff' of complexity with Mockaroo: For example, if they couldn't express the intended relationship between arrival and departure times, they settled for uniformly random times for both. Of the six users that spent less than 5 minutes specifying the AH task on Mockaroo, four did not even attempt to formulate a relationship between age and height and specified height as a random value within a fixed range. One user incorrectly declared height = age×2. Note that the relationship between height and age is roughly linear (with minimal tapering between 15 and 16) and it can be directly estimated from the growth charts as height = age×6 + 75, with some added random noise, and some shifts introduced to account for gender differences.

Dataset Realism

The Checkers. To evaluate how real the generated datasets were, we recruited three new participants (all undergraduates). We provided them with two randomly selected dataset samples of 1000 data points from each task, as well as the titles of the dataset: Birth Records, Train Schedules from western Europe, and Paediatric Age and Height Records. We asked the participants to help us determine if these datasets *could* be real or fake. We asked them to explain their conclusions and, in plain English, write checks that we could use for other, similar samples that they have not seen. We used the samples to ground their reasoning as well as to eliminate simplistic checks like 'age should be a number'. We encouraged the participants to use any data analysis tools to help them with the process, but we provided the data in Excel spreadsheets. We observed that users often created visualizations in Excel to examine gender or country distributions. We recruited external checkers that did not generate the dataset to eliminate any biases introduced by working through the data generation process.

[6]Mauchly's test of sphericity indicated that the assumption of sphericity was met.

We coded the responses to find common rules from the checks. Each checker provided 3 to 5 checks per task. We eliminated conflicting checks. For example, one checker wanted trains to depart or arrive uniformly across all hours of the day; another checker felt that trains should mostly depart during peak hours. We eliminated checks that required external datasets that were not available to the developers or users during data generation. For example, two checkers wanted us to verify that travel times were positively correlated with distance between cities.

Table 2 summarizes the final set of rules that we evaluated. Using these rules as a metric for how real a dataset is, we found that datasets generated with Synner usually fare better than those generated by developers or with Mockaroo.

We now explore these results in the context of some of the technical challenges and design principles discussed earlier.

Correctly implementing a statistical distribution. We observed that developers were prone to make errors even with simple statistical distributions. This impacted realism with respect to rules 3 and 10. For example the following code snippet, accidentally results in a 2:1 distribution of women to men: The developer assumed that the random integer generator only returns '-1' or '1' and did not consider '0'.

```
genderInd = random.randint(-1,1)
gender = "F"
if genderInd == 1:
    gender = "M"
```

Instant visual feedback by providing a histogram of gender would have helped the developer detect the problem.

Correctly implementing relationships. In Mockaroo, the order of specifying fields usually determines the order of data generation and users have to specify independent fields before dependent ones[7]. These ordering rules, however, do not apply when generating data jointly from domains with natural dependencies such as cities and countries. For rule 1, users had to specify the parent's last name first for the child's last name to depend on it. When expressing the relationship failed because of incorrect ordering, eight users simply gave up and generated the last name randomly. These ordering semantics can have insidious effects: for example in Mockaroo's domain database, there are roughly 70 French cities and several western European countries with only a handful of cities: this resulted in an unusual distribution of countries in the TS datasets generated with Mockaroo: France was in about 80% of all the train trips. Reordering columns such that country occurs before city does not impact the data generation order. Synner does not require users to add fields in a specific order to control dependencies and the semantics of dependencies are consistent. The declarative, visual specification means that users can clearly specify how fields depend on each other, and Synner implements the correct data generation order that matches the user specification. If users declared that city depends on country (as all users did in Synner) then countries were sampled uniformly and each generated country probed the relationship join of cities and countries to generate a city.

[7]We thoroughly covered order semantics in Mockaroo's tutorial

Task	#	Data is real if …	Evaluated On	Synner	Mockaroo	Developers	Freelancer	CS Grads
Birth Records	1	Most children have the same last name as one of their parents last names	DS	0.78	0.56	0.80	1.00	0.60
	2	Names are gender-typical	C	0.89	0.91	0.87	0.93	0.80
	3	The distribution of males and females is uniform	DS	1.00	1.00	0.70	0.60	0.80
	4	There are no unusually frequent name patterns	DS	1.00	1.00	0.80	0.60	1.00
Train Schedules	5	Arrival time is at least 15 minutes after departure time	R	0.81	0.68	0.69	0.77	0.62
	6	There are no unusually frequent countries	DS	1.00	0.00	0.80	0.80	0.80
Pediatric Age & Height	7	On average, boys are taller than girls	DS	0.83	0.61	0.90	1.00	0.80
	8	For each age group, there is some height variance	R	0.98	0.62	0.93	1.00	0.85
	9	Age and Height follow the appropriate distributions for children*	DS	1.00	0.67	0.90	1.00	0.80
	10	The distribution of males and females is uniform	DS	1.00	1.00	0.70	0.60	0.80
	11	Age is less than or equal to 16 years old	DS	1.00	1.00	0.90	0.80	1.00

Table 2. For each check, we report the proportion of datasets that satisfy the rule if the rule was evaluated on the entire data set (DS). For rules evaluated on columns (C), we report the proportion of columns across all datasets that satisfy the rule. For example, there are three name columns per BR dataset for which rule #2 applies. For rules evaluated on rows or row groups (R), (e.g. AH rows grouped by age), we report the proportion of rows or groups that satisfy the rule. *Further clarifications on how rule 9 was implemented are detailed in the text.

The reverse dependency, country depends on city, would have resulted in cities controlling the distribution of countries.

To evaluate whether age and height fit the CDC children's growth charts: we generated an ideal sample using the growth charts. We then computed the Kullback-Leibler divergence, which measures how one probability distribution is different from another. Figure 10 illustrates the KL-divergence presented as the percentage increase in bits required to encode the ideal sample's distribution with a given sample's distribution. If we generated a sample of heights uniformly distributed between 80 and 200 cm, independent of age, we would require an extra 15.6% bits to encode the ideal sample. Using this as a weak threshold for how real the age and height samples are, we present in Table 2 the proportion of samples that are better than random. In Figure 10, the four Mockaroo samples that were worse than a random sample achieved so by expressing an incorrect linear relationship between age and height (height = age × 40; age × 20; age × 10 or age × 2). As we mentioned earlier, users often hit a cliff of complexity when trying to express the relationship between age and height. Even when users were able to express the relationship, they often had difficulties adding some variance (rule 8) and ended up with a specific height for a given age. We noticed that developers employed a variety of strategies to generate this dataset: some created a look-up table of height ranges for each age value — a height was selected uniformly at random from this range, others created a straight line equation with a slope and an intercept that they estimated from the growth charts. One developer generated height randomly and independently of age.

Specifying the relationship between age and height by drawing it in Synner, and visually controlling the degree of Gaussian noise with a slider, drastically simplified the process. We did observe, however, that users overemphasized the tapering in height that occurs around 15 years old, and tapered height much earlier at around 10 or 11 years old. These results resonate with recent research on how users (mis)sketch time-series patterns [23]. In the future we intend to annotate such inflection points to allow users to realize such sketching errors and correct them.

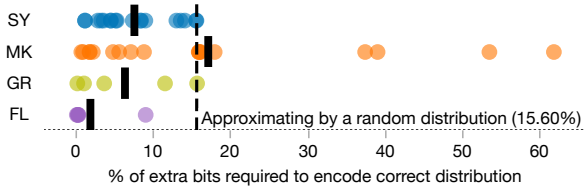

Figure 10. For the Age & Height pediatric records data set, we compute the KL-divergence of the generated sample from an ideal sample derived from the CDC male and female growth charts (age groups 2-16). From the KL-divergence, we determine the percentage increase in bits required to approximate the ideal distribution by the generated sample's distribution. For reference, if the sample's distribution is completely random, then we can approximate the ideal distribution with a 15.6% increase in bits. A sample perfectly represents the ideal sample if it needs 0% extra bits.

Qualitative Evaluations

We also administered a post-study questionnaire to gain some qualitative feedback on Synner. On a 5-point Likert scale, we asked users to rate each tool's ease of use, expressiveness or power, and realism of the generated data. Overall users found both Synner and Mockaroo easy to use ($\mu_{SY} = 4.2$, ; $\mu_{MK} = 4.1$,). They found Synner to be more expressive and more powerful when compared to Mockaroo, $\mu_{SY} = 4.6$, ; $\mu_{MK} = 3.6$, , and also more capable of generating realistic data, $\mu_{SY} = 4.6$, ; $\mu_{MK} = 3.9$, .

We asked users to elaborate on their ratings. Many users commented on the numerous domains available in Mockaroo and wished that we expanded the available domains in Synner: *"I preferred that [Mockaroo] had a bigger menu range for choosing different types of data"*. A number of users commented that for simple tasks, Mockaroo was lot more easy to use than Synner but as the tasks became more complex, the reverse was true: *"[Mockaroo] is useful for very simple data and if*

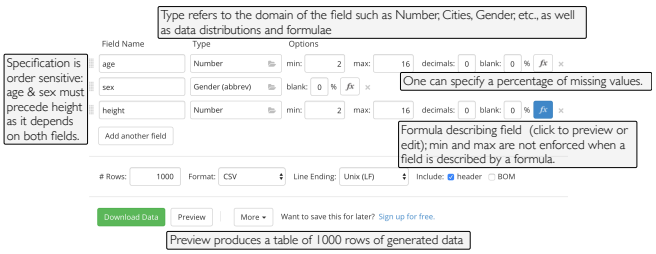

Figure 11. A possible specification of the pediatric age and height records data set in Mockaroo.

the user doesn't care much about details. However, it doesn't provide the user with lots of options to manipulate the data," "[Mockaroo] gets worse the more complicated and inter-related the data is," "[Mockaroo is] not really easy. I spent 11 min trying to generate if-else functions and struggled with it," "Relationships happen behind the scenes [in Mockaroo] and if it does not happen automatically then there is no way to define relationships," "It is very good at relating fields automatically, but [Mockaroo] gives little freedom beyond that."

Many users appreciated Synner's example-driven interaction features. One user noted that it contributed to the tool's overall ease of use: *"The auto-suggestions and fills were very useful."*. For each task, we counted the number of times users used Synner's example-driven interaction features: For the BR task, users used these features an average of 7 times out of 8 possible opportunities; For TS task, 5.2 out of 6; For the AH task, 4.3 out of 5.

Synner's interactive visualizations of generated data, ability to communicate randomness and expressive power might have increased user trust in the overall realism of the generated datasets: *"I think most of the data was realistic and it was easy [in Synner] to make adjustments too if something didn't seem right;" "In the examples I did, the data generated was pretty realistic: [Synner] allows you to visually represent it so you can get more accurate results;" " you can make sure that each field is consistent with the others; especially with the ability to control distributions and correlations at such a granular level."* This contrasted with some of the users' comments on their experiences using Mockaroo: *"Less control over data generated and quality of data produced was questionable ... clicking preview each time was a hassle and it wasn't clear whether a [relationship] existed or not," "I had to manually count numbers as I changed data parameters," "You couldn't see the preview until the end, you didn't have any idea what the output would be until then", "It was difficult because you couldn't be sure whether you had done something correctly without clicking on the preview button."*

RELATED WORK

We first describe Mockaroo, a commercial, and user-friendly tool for realistic data generation, that requires no programming [2]. Figure 11 illustrates how one can generate the pediatric age and height records with Mockaroo. A simple form-based interface and an extensive library of domains allows users to quickly generate realistic data with straightforward relationships. Mockaroo provides some support for custom distributions over discrete values with basic support for case dependencies[8]. This feature does not extend to continuous distributions. The following interface design limitations of Mockaroo make the specification of complex data generation tasks difficult: (i) Mockaroo is order-sensitive, which means its specification interface is not entirely declarative. In Figure 11, height has to occur after age and sex to allow height to depend on both. When a user's specification order is different from the partial order imposed by dependencies, debugging the source of the data generation errors can be frustrating. Missing values can also cause cascading errors. In Figure 11, a missing age value results in an error message to be generated for the corresponding height. (ii) Mockaroo lacks any form of visual specification support. Thus users have to know a priori the exact statistical distributions for their data and how independently distributed fields affect dependents. (iii) Mockaroo provides no interactive, visual previews. This makes it difficult for users to validate the correctness or realism of their specifications, or to easily refine the parameters of their distributions.

Our work stands apart from synthetic data generators for testing code [9]. It is more closely related to synthetic data generation for database benchmarking or SQL testing [26, 28, 15, 5, 29, 11]. There are several points of departure, however, from this body of work.

First, Synner does not assume access to a real dataset (or a real database schema and meta-data which contains table histograms and join selectivity stats) from which we can estimate a multidimensional (statistical) model that allows us to generate datasets of any scale like the following systems: [26, 28, 15]. Like Synner, these works do strive for data realism: they infer as accurately as possible the data distributions of the provided sample dataset. Unlike, Synner, they do not help users build datasets from scratch and, in a tight human-in-the-loop fashion, specify and visually validate their specifications.

Second, Synner's underlying data generation language differs from other data generation languages: It is declarative unlike these works: [11, 5]. It is also deliberately not as complex: when testing SQL queries or features of a database management system like a new index structure, one needs to create datasets that meet exact cardinality or aggregation constraints [29, 6] such as the size of the result set for a specific query needs to be bigger than x or smaller than the result set of another query, or that the average of a set of tuples should be more than the average of another (overlapping) set of tuples. While Synner's language allows us to generate datasets where men on average are taller than women, the language does not enforce such a guarantee (there is a small statistical chance that we end up with a generated dataset with no difference in means) and this is communicated visually. More importantly, Synner allows users to specify different distributions for men and women to achieve this property rather than specify the property on averages and utilize solvers or other heuristics to automatically infer a valid distribution for each group [29]. Thus, Synner's language sits at a different

[8] https://mockaroo.com/help/list

point in the power-usability tradeoff from these data generation languages: we deliver more expressive power than tools like Mockaroo but are more user-friendly than these data generation scripting languages. Like many of these languages, we require that dependencies form a DAG to ensure tractable data generation [29].

In Synner, relationships between fields form a DAG of dependencies. When these dependencies are conditional probability distributions, such as when the mean of a normal distribution for height depends on sex, then the data generation model is effectively a *Bayesian model*. Tools like JAGS (Just Another Gibbs Sampler) [25] and the different variants of BUGS (Bayesian inference Using Gibbs Sampling) [22] support the simulation of data from a Bayesian model, which can be specified either through code or visually through a graph. The primary purpose, however, of these tools is to infer the parameters of prior distributions given input data. Simulation experts who would like to test the validity of a hypothesized model for a given data set often use these tools. Synner does not support inference. It, however, enables realistic data generation by a much larger user base than simulation experts by (i) allowing users to iteratively and interactively specify data generation tasks with little statistical or Bayesian modelling expertise, (ii) by providing richer domains and support for domain relationships, and (iii) by allowing visual and example-driven specifications of fields.

Synner's design principles of visual lifting, declarative specification, and example-driven interaction were influenced by Heer et al.'s work on predictive interaction for data transformation [13] and Horvitz's design principles for Mixed-initiative user-interfaces [14].

Our strategies for communicating uncertainty in the outcomes of a data generation process are influenced by the rich literature on uncertainty visualization [16]. Like Hullman's hypothetical outcome plots, our Layers visualization generates multiple outcomes of the generation process and visualizes them [17]. Synner has the added challenge of integrating the visual specification of the data distribution along with the uncertainty associated with the data generation process.

CONCLUSION

In this paper, we described the technical challenges of generating realistic data, and the design principles of a powerful and expressive, yet easy to use data generation tool. We built Synner following these principles and conducted user studies to evaluate how effective Synner is at generating data and how realistic datasets generated with Synner are. We find that on average users spend less time specifying more complex data generation tasks with Synner than with Mockaroo: a popular and publicly accessible data generation tool. We also find than on user-provided checks of data realism, datasets generated with Synner satisfy more checks than datasets generated by developers or with Mockaroo.

Currently, Synner's visual interface can easily handle datasets with thousands of records: interactivity degrades, however, beyond that. While the downloaded data generation script can produce orders of magnitude more data when executed in Synner's command-line, interactivity and uncertainty visualization at massive data scale is a limitation we intend to address in the future, perhaps with visualization-aware sampling techniques [20, 24]. While this limitation may not impact pedagogical and software-demo use cases, it limits our ability to visually understand (and hence visually specify) the behavior of different data distributions and relationships at massive scale.

Finally, we chose the name Synner as a portmanteau for '*Syn*thetic data generat*or*' but also as it is a homophone for sinner. We are aware of the implications of making realistic data generation easy and we are looking for ways to prevent misuse. We believe that our tool will benefit many users including members of the UIST community who may wish to create compelling demonstrations, or interesting datasets for class projects. At the moment, we have weak safeguards in place such as purely statistical data generation, which means that the same script is unlikely to yield the same dataset; and what you see in the spreadsheet is not what you download. We will also open-source the project.

REFERENCES

[1] 2018. Gallup World Poll. (2018). https://www.gallup.com/analytics/232838/world-poll.aspx

[2] 2019. Mockaroo: Random Data Generator. https://www.mockaroo.com/. (2019).

[3] Jad Abumrad and Robert Krulwich. 2009. A Very Lucky Wind. Podcast produced by WNYC Studios. (June 2009). https://www.wnycstudios.org/story/91686-a-very-lucky-wind

[4] Ken Brodlie, Rodolfo Allendes, Osorio, and Adriano Lopes. 2012. *A Review of Uncertainty in Data Visualization*. Springer London, London, 81–109. DOI: http://dx.doi.org/10.1007/978-1-4471-2804-5_6

[5] Nicolas Bruno and Surajit Chaudhuri. 2005. Flexible Database Generators. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB '05)*. VLDB Endowment, 1097–1107. http://dl.acm.org/citation.cfm?id=1083592.1083719

[6] Nicolas Bruno, Surajit Chaudhuri, and Dilys Thomas. 2006. Generating Queries with Cardinality Constraints for DBMS Testing. *IEEE Trans. on Knowl. and Data Eng.* 18, 12 (Dec. 2006), 1721–1725. DOI: http://dx.doi.org/10.1109/TKDE.2006.190

[7] Hadley Cantril. 1966. *The pattern of human concern.* Rutgers University Press.

[8] GAISE College Report ASA Revision Committee. 2016. Guidelines for Assessment and Instruction in Statistics Education College Report 2016. (2016). http://www.amstat.org/education/gaise

[9] Jon Edvardsson. 1999. A survey on automatic test data generation. In *Proceedings of the 2nd Conference on Computer Science and Engineering*. 21–28.

[10] FiveThirtyEight. 2019. Data and code behind the articles and graphics at FiveThirtyEight. (2019). https://data.fivethirtyeight.com/

[11] Jim Gray, Prakash Sundaresan, Susanne Englert, Ken Baclawski, and Peter J. Weinberger. 1994. Quickly Generating Billion-record Synthetic Databases. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data (SIGMOD '94)*. ACM, New York, NY, USA, 243–252. DOI: http://dx.doi.org/10.1145/191839.191886

[12] Rui Han, Zhen Jia, Wanling Gao, Xinhui Tian, and Lei Wang. 2015. Benchmarking Big Data Systems: State-of-the-Art and Future Directions. *CoRR* abs/1506.01494 (2015). http://arxiv.org/abs/1506.01494

[13] Jeffrey Heer, Joseph Hellerstein, and Sean Kandel. 2015. Predictive Interaction for Data Transformation. In *Conference on Innovative Data Systems Research (CIDR)*. http://idl.cs.washington.edu/papers/predictive-interaction

[14] Eric Horvitz. 1999. Principles of Mixed-initiative User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '99)*. ACM, New York, NY, USA, 159–166. DOI: http://dx.doi.org/10.1145/302979.303030

[15] Kenneth Houkjaer, Kristian Torp, and Rico Wind. 2006. Simple and Realistic Data Generation. In *Proceedings of the 32Nd International Conference on Very Large Data Bases (VLDB '06)*. VLDB Endowment, 1243–1246. http://dl.acm.org/citation.cfm?id=1182635.1164254

[16] J. Hullman, X. Qiao, M. Correll, A. Kale, and M. Kay. 2019. In Pursuit of Error: A Survey of Uncertainty Visualization Evaluation. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (Jan 2019), 903–913. DOI: http://dx.doi.org/10.1109/TVCG.2018.2864889

[17] Jessica Hullman, Paul Resnick, and Eytan Adar. 2015. Hypothetical Outcome Plots Outperform Error Bars and Violin Plots for Inferences About Reliability of Variable Ordering. *PLOS ONE* 10, 11 (2015). http://idl.cs.washington.edu/papers/hops

[18] Andrew T Jebb, Louis Tay, Ed Diener, and Shigehiro Oishi. 2018. Happiness, income satiation and turning points around the world. *Nature Human Behaviour* 2 (2018), 33–38. Issue 1. DOI: http://dx.doi.org/10.1038/s41562-017-0277-0

[19] Matthew Kay, Tara Kola, Jessica R. Hullman, and Sean A. Munson. 2016. When (Ish) is My Bus?: User-centered Visualizations of Uncertainty in Everyday, Mobile Predictive Systems. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 5092–5103. DOI: http://dx.doi.org/10.1145/2858036.2858558

[20] Albert Kim, Eric Blais, Aditya Parameswaran, Piotr Indyk, Sam Madden, and Ronitt Rubinfeld. 2015. Rapid Sampling for Visualizations with Ordering Guarantees. *Proc. VLDB Endow.* 8, 5 (Jan. 2015), 521–532. DOI: http://dx.doi.org/10.14778/2735479.2735485

[21] Albert Kim, Chester Y.Ismay, and Jennifer Chunn. 2018. The fivethirtyeight R Package: "Tame Data" Principles for Introductory Statistics and Data Science Courses . *Technology Innovations in Statistics Education* 11 (2018). Issue 1. https://escholarship.org/uc/item/0rx1231m

[22] David J. Lunn, Andrew Thomas, Nicky Best, and David Spiegelhalter. 2000. WinBUGS — A Bayesian Modelling Framework: Concepts, Structure, and Extensibility. *Statistics and Computing* 10, 4 (Oct. 2000), 325–337. DOI: http://dx.doi.org/10.1023/A:1008929526011

[23] Miro Mannino and Azza Abouzied. 2018. Expressive Time Series Querying with Hand-Drawn Scale-Free Sketches. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 388, 13 pages. DOI: http://dx.doi.org/10.1145/3173574.3173962

[24] Y. Park, M. Cafarella, and B. Mozafari. 2016. Visualization-aware sampling for very large databases. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. 755–766. DOI: http://dx.doi.org/10.1109/ICDE.2016.7498287

[25] Martyn Plummer and others. 2003. JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling. In *Proceedings of the 3rd international workshop on distributed statistical computing*, Vol. 124. Vienna, Austria., 10.

[26] Tilmann Rabl, Manuel Danisch, Michael Frank, Sebastian Schindler, and Hans-Arno Jacobsen. 2015. Just Can'T Get Enough: Synthesizing Big Data. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*. ACM, New York, NY, USA, 1457–1462. DOI: http://dx.doi.org/10.1145/2723372.2735378

[27] Dan Rinzel. 2016. Staging to Sell: the seduction & danger of demo data. (2016). https://medium.com/@dan2bit/staging-to-sell-the-seduction-danger-of-demo-data-9a229981d32 Last Accessed on March 29, 2019.

[28] Entong Shen and Lyublena Antova. 2013. Reversing Statistics for Scalable Test Databases Generation. In *Proceedings of the Sixth International Workshop on Testing Database Systems (DBTest '13)*. ACM, New York, NY, USA, Article 7, 6 pages. DOI: http://dx.doi.org/10.1145/2479440.2479445

[29] Emina Torlak. 2012. Scalable Test Data Generation from Multidimensional Models. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE '12)*. ACM, New York, NY, USA, Article 36, 11 pages. DOI: http://dx.doi.org/10.1145/2393596.2393637