Synner: Generating Realistic Synthetic Data

Miro Mannino
NYU Abu Dhabi
miro.mannino@nyu.edu

Azza Abouzied
NYU Abu Dhabi
azza@nyu.edu

ABSTRACT

SYNNER allows users to generate realistic-looking data. With SYNNER users can visually and declaratively specify properties of the dataset they wish to generate. Such properties include the domain, and statistical distribution of each field, and relationships between fields. User can also sketch custom distributions and relationships. SYNNER provides instant feedback on every user interaction by visualizing a preview of the generated data. It also suggests generation specifications from a few user-provided examples of data to generate, column labels and other user interactions. In this demonstration, we showcase SYNNER and summarize results from our evaluation of SYNNER's effectiveness at generating *realistic-looking* data.

CCS CONCEPTS

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

KEYWORDS

data generation, declarative languages, example-driven interaction

ACM Reference Format:
Miro Mannino and Azza Abouzied. 2020. Synner: Generating Realistic Synthetic Data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3318464.3384696

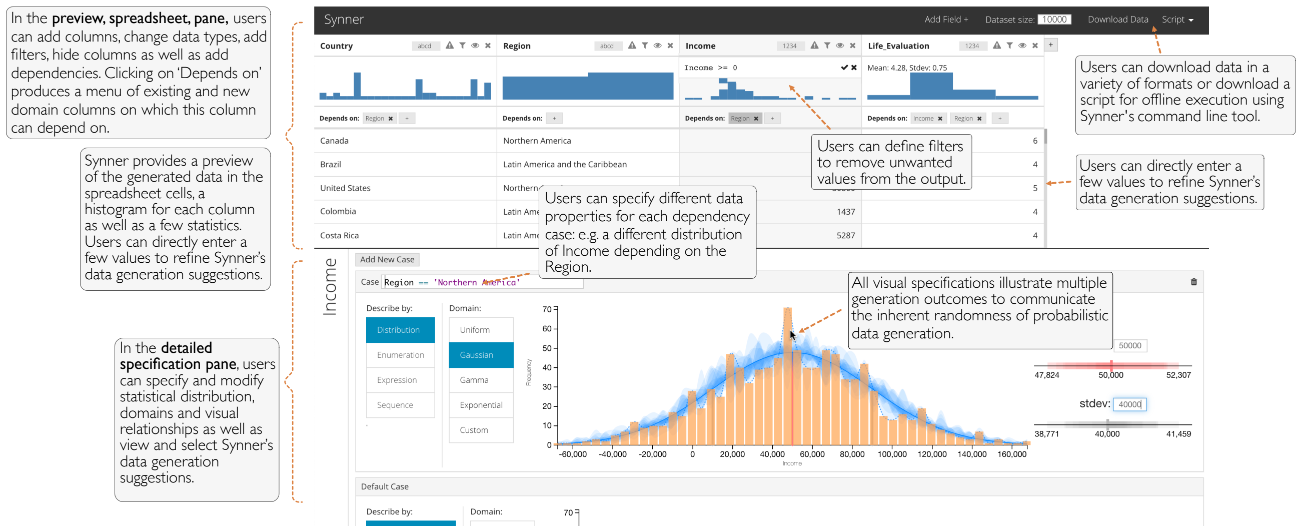
1 INTRODUCTION

For many users from data scientists, to educators, to software testers, data generation is important. Often real datasets are (a) inaccessible such as private medical or financial records, (b) expensive such as twitter's data stream, (c) too coarse grained such as energy consumption records aggregated by district instead of by household, or (d) incomplete e.g. the data is missing important fields, or is a small subset of the entire dataset. In these situations, we resort to synthetic data as a functional alternative to real data.

However, generating realistic synthetic data is technically challenging, especially since only a handful of tools support basic data generation through a user-friendly UI (e.g. MOCKAROO [1]) . As we empirically demonstrate in our full paper describing SYNNER [4], 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. There are several reasons why data generation is a technically complex task: First, generating realistic data requires a fair understanding of statistical modeling and developers often have to correctly transform standard distributions to custom ones or identify appropriate parameters for distributions through trail and error. Second, dependencies or relationships between fields often introduce ordering constraints on how we generate data and if these are not implemented correctly, data generation can become computationally intractable or produce unreal data. Finally, scaling down a dataset can result in the biased sampling of domains and scaling up a dataset can result in unintended repetitions or poor performance.

For many users who view data generation as secondary albeit essential to their primary task, investing the time and resources to generate the right data is often unjustifiable. To address these challenges, we designed and built SYNNER with the following principles:
Visual Lifting: SYNNER generates data from scripts in a custom declarative *data generation language* (See Listing 1). SYNNER visually lifts from these scripts to a more intuitive visual domain (See Figure 1). It generates, samples and presents data at every user interaction in a familiar spreadsheet visualization.

The interface guides users on how to add fields, dependencies and even what distributions to use or which domains to pick data from. Users can even draw custom univariate distributions or bivariate relationships.
Declarative Specification. Users only focus on what the data looks like but not how it is generated: they do not specify the order of generating data 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



In the **preview, spreadsheet, pane,** users can add columns, change data types, add filters, hide columns as well as add dependencies. Clicking on 'Depends on' produces a menu of existing and new domain columns on which this column can depend on.

Synner provides a preview of the generated data in the spreadsheet cells, a histogram for each column as well as a few statistics. Users can directly enter a few values to refine Synner's data generation suggestions.

In the **detailed specification pane**, users can specify and modify statistical distribution, domains and visual relationships as well as view and select Synner's data generation suggestions.

Users can download data in a variety of formats or download a script for offline execution using Synner's command line tool.

Users can directly enter a few values to refine Synner's data generation suggestions.

Users can define filters to remove unwanted values from the output.

Users can specify different data properties for each dependency case: e.g. a different distribution of Income depending on the Region.

All visual specifications illustrate multiple generation outcomes to communicate the inherent randomness of probabilistic data generation.

Figure 1: Synner's user interface

statistical distribution, or how to jointly generate values from related domains such as city and country.

Example-driven Interaction. Synner infers what users wish to generate from column labels or values manually inserted into the data spreadsheet. It suggests an ordered list of the top five generator recommendations. It also suggests dependencies or relationships between domains.

2 DEMONSTRATION

At SIGMOD 2020, we will demonstrate Synner's ability to effectively generate synthetic data. Attendees can use Synner to design their own synthetic data sets through two laptops pre-loaded with Synner or by accessing our online demo at synner.io. We will guide attendees through the main interface features of Synner as well as describe Synner's underlying data generation engine and its many optimizations, which would be of interest to the data management community. While attendees can make their own datasets, we will showcase Synner through a guided use-case scenario.

Guided Demo Scenario: *The Economics Professor.* Marcia is an Economics professor who teaches at a small college in Latin America. She generates a dataset to allow her first-year quantitative methods students to analyze the relationship between happiness and income across Northern America, and Latin America & the Caribbean. She bases her generated dataset on the results of a recent study that explores income and happiness across the world [3] and uses the Gallup World Poll dataset, which is not publicly available. She wishes to generate roughly 10,000 data points. She hopes her students would analyze and visualize the dataset, which

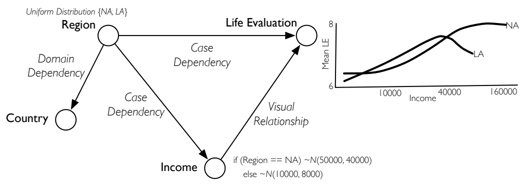


Figure 2: The properties of Marcia's desired synthetic dataset.

she explains illustrates the relationship between income and happiness and differences across regions of varying wealth. She sketches the data schema in Figure 2.

Figure 1 illustrates the different features of Synner and the final preview after Marcia generates her entire dataset, with the focus on the Income column. To generate her data, Marcia simply begins by adding a new column. As soon as she labels it Country, Synner suggests using a Country domain generator in the details pane. On selecting that generator, the preview column is populated with a few randomly selected countries. A histogram at the top of the column shows the distribution of different countries. She wishes Country to depend on Region so she clicks *depends on*, which causes Synner to present a drop-down menu of fields to select or even create that Country can depend on. Using a backend domain database that contains data of known domains such as City, Name, Continent, Region, Zipcodes, etc. and pairwise relationships between related fields, Synner suggests new fields such as City, Region, Zipcodes, etc., that can be related with Country through a minimal join path of relationship tables. Marcia selects Region, which creates a new column labeled Region with values randomly selected from the Region domain. Now each row shows a country within

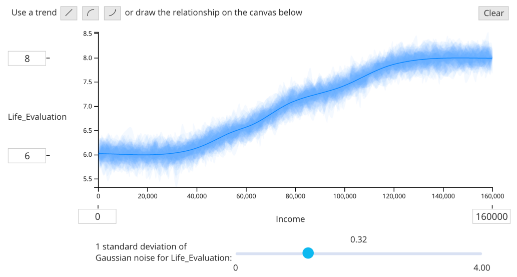


Figure 3: Visual relationship between Income and Life Evaluation when Region is Northern America.

the given region. Marcia can now filter values in the Region to only those in 'Northern America' and 'Latin America and the Caribbean'.

She then creates the Income field and types in a few values into the column. Synner suggests several statistical distributions that fit the examples given. She selects a normal distribution and sets the mean to 10000 USD and the standard deviation to 8000 USD, which roughly matches the statistics on income in Latin America in her reference study [3]. Since she wants a different income distribution for North America, she adds Region as a dependency, which allows her to "Add a New Case" (See Figure 1). She then creates a different normal income distribution for rows where Region == 'Northern America': mean income of 50000 USD and standard deviation of 40000.

Finally, she creates the Life Evaluation field. Life Evaluation is a self-reported 0-10 score on Cantril's Self-Anchoring Striving Scale [2]. Using the plot of life evaluation against income by world regions from the study (Figure 2), Marcia declares that Life Evaluation depends on Income and specifies two visual relationships between income and life evaluation, one for the Northern America region (Figure 3) and one for Latin America & the Caribbean. She also controls how much noise is added to the relationship.

Through this small but realistic scenario, we demonstrate most of Synner's features and design principles, which are discussed in detail in the full paper [4]. We can easily add more fields with different properties and relationships to illustrate and clarify Synner's data generation power as well as demonstrate features not in described in this paper for brevity such as Synner's ability to introduce errors and missing values, and Synner's data visualizations that aim to communicate the randomness of probabilistic data generation [4].

2.1 The Data Generation Engine

Every user interaction that changes the properties of the data set automatically updates a declarative data generation script. The script describes the properties of each field such as its domain, its statistical distribution, and its relationships

```
{ "model": {                                                  1
  "Country": {
    "type": "string",                                         3
    "dependencies": ["Region"],
    "generator": {"domain": "COUNTRY"}},                      5
  "Region": {
    "type": "string",                                         7
    "filter": "Region == 'Northern America' ||
              Region == 'Latin America'",                     9
    "generator": {"domain": "REGION"}},
  "Income": {                                                 11
    "type": "integer",
    "dependencies": ["Region"],                               13
    "generator": {
      "switch": [{                                            15
        "case": "Region == 'Northern America'",
        "then": {                                             17
          "distribution": "normal",
          "mean": 50000, "stdev": 40000}},{                   19
        "default": {
          "distribution": "normal",                           21
          "mean": 10000, "stdev": 8000}}]}},
  "Life_Evaluation": {                                        23
    "type": "integer",
    "dependencies": ["Income", "Region"],                     25
    "generator": {
      "switch": [{                                            27
        "case": "Region == 'Northern America'",
        "then": {                                             29
          "visrel": {
            "x-field": "Income",                              31
            "x-sketch-normalized": [0, ..., 1],
            "y-sketch-normalized": [0, ..., 0.994],           33
            "x-min": 0, "x-max": 160000,
            "y-min": 6, "y-max": 8,                           35
            "noise": 0.0552}}},{
        "default": {...}}]}}}},                               37
"data": {
    "size": 10000, ...                                        39
}}
```

Listing 1: The Data Generation Script for Marcia's dataset

to other fields, as well as overall properties of the data to be generated (e.g. the dataset size). Hand-sketched custom distributions or visual relationships are encoded as normalized points on a two-dimensional unit square (See Listing 1, lines 33-36).

The engine *parses* the script to create an *optimal, executable, data generation pipeline*, which it executes. The engine returns a sample of the data to the front-end user interface for preview as well as different summary statistics, but stores the entire generated dataset. This can later be downloaded in different formats.

2.1.1 Parser. The parser translates the data generation script to a directed acyclic graph (DAG). For each field a node is created, with directed edges representing relationships such as domain dependencies, case dependencies or visual relationships between fields. For example, Figure 2 illustrates the DAG the parser produces from Marcia's dataset specification. Since the DAG determines the data generation order, no cycles are allowed.

2.1.2 Planner & Optimizer. The planner topologically sorts the nodes of the DAG to produce a data generation order.

The script in listing 1 results in the following data generation order: [1: {Region}, 2: {Country, Income}, 3: {Life Evaluation}].

Synner always pushes down selection operations, i.e. it applies them at the field with the filter condition, immediately after its value is generated, and not post-hoc after the entire data set is generated. The optimizer can choose to apply field-level selection operations to the plan as follows:

(1) *Block until valid.* If the generated value violates the selection condition, a new value is repeatedly generated until one satisfies the condition,
(2) *Start fresh.* If the generated value violates the selection condition, the entire tuple is deleted, or
(3) *Further push-down.* If the field depends on other base fields, the optimizer attempts to create new selection operations for these base fields.

The optimizer opts for *Start fresh* or *Further push-down* if the filtered field depends on other base fields in such a way that a value cannot be generated that satisfies the selection condition given the values of the base fields.

2.1.3 Scheduler. Given the data generation plan, the scheduler executes a sequence of *generators* in order of the plan. Each generator looks up values that it depends on from a global dataset table, generates its value, and updates (or deletes) the appropriate row in the table. Synner supports the following generators (and compositions of them):

(1) *Statistical generators*, which sample from many standard, parameterizable, distributions such as normal, uniform, gamma, exponential, etc as well as custom distributions.
(2) *Expression generators*, which allow arbitrary expressions over fields and string regular expressions.
(3) *Enumeration generators*, which sample from a user-provided list of values with different frequencies.
(4) *Domain generators*, which sample from domains stored in the domain database. Currently Synner supports a few domains, which we hope to expand in the future.
(5) *Sequence generators*, which generate deterministic sequences of values e.g. 1, 2, 3, ... or 10:00, 10:15, ... etc.
(6) *Case generators*, which switch between different generators depending on which case, defined by a predicate on one or more previously generated fields, is satisfied. A default case generator must always be defined.
(7) *Visual-Relationship generators*, which define a visual relationship between a pair of fields (e.g Figure 3). Visual relationships can also be configured with some Gaussian noise.
(8) *Domain-Dependency generators*, which sample data from one or more joined relationship tables in the domain database. Synner searches for the minimal join path between two domain-dependent fields that have no direct relationship table. For example, in our domain database, there is no relationship table between City and Region, but the database contains these two relationship tables Relationship.City_Country and Relationship.Country_Region. In order to generate values for City that depend on Region Synner samples from the join of the two tables: Relationship.City_Country $\bowtie$ Relationship.Country_Region.

In the demonstration, we will also discuss our experiences with several experimental optimizations including (a) efficient statistical data generation by pre-computing and materializing columns of random values (b) utilizing column-store layouts for efficient incremental data generation, (c) efficient join materialization and sampling techniques, and (d) different distributed processing techniques.

2.2 Evaluation Summary

We conclude our demonstration by describing the two main takeaways from our evaluation of Synner's user interface [4]: First, users complete more complex tasks in significantly less time with Synner than with Mockaroo— a popular, easy-to-use data generation tool. Second, users generated more realistic data sets with Synner than with alternatives (including custom scripts written by hired developers) as judged by an external group of realism checkers. Details of the different experiments and our analysis can be found in the main publication [4].

3 CONCLUSION

Our goal in this demonstration is to illustrate how an interactive, example-driven interface that visually lifts from a declarative data generation language and is backed by a modular data generation engine can help users easily and effectively generate rich and realistic synthetic data sets. In our demonstration, attendees can experience for themselves the power of Synner's design. We also describe our on-going efforts to improve the functionality and scalability of Synner's data generation system through expanding the library of generators, the domain database and our existing and experimental implementations of a variety of optimizations such as selection push-down, efficient join sampling, etc.

REFERENCES

[1] 2019. Mockaroo: Random Data Generator. https://www.mockaroo.com/.
[2] Hadley Cantril. 1966. *The pattern of human concern.* Rutgers University Press.
[3] 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. https://doi.org/10.1038/s41562-017-0277-0
[4] Miro Mannino and Azza Abouzied. 2019. Is This Real? Generating Synthetic Data That Looks Real. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 549–561. https://doi.org/10.1145/3332165.3347866