The Synthetic Data Vault: Generative Modeling for Relational Databases

by

Neha Patki

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

Author .....
Department of Electrical Engineering and Computer Science
May 18, 2016

Certified by .....
Kalyan Veeramachaneni
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Dr. Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee

The Synthetic Data Vault: Generative Modeling for Relational Databases

by

Neha Patki

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 2016, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

The goal of this thesis is to build a system that automatically creates synthetic data for enabling data science endeavors. To meet this goal, we present the Synthetic Data Vault (SDV), a system that builds generative models of relational databases. We are able to sample from the model and create synthetic data, hence the name SDV.

When implementing the SDV, we developed an algorithm that computes statistics at the intersection of related database tables. We then use a state-of-the-art multivariate modeling approach to model this data. The SDV iterates through all possible relations, ultimately creating a model for the entire database. Once this model is computed, the same relational information allows the SDV to synthesize data by sampling from any part of the database.

After building the SDV, we used it to generate synthetic data for five different publicly available datasets. We then published the datasets and asked data scientists to develop predictive models for them as part of a crowdsourced experiment. On May 18, 2016, preliminary analysis from the ongoing experiment provided evidence that the synthetic data can successfully replace original data for data science. Our analysis indicates that there is no significant difference in the work produced by data scientists who used synthetic data as opposed to real data. We conclude that the SDV is a viable solution for synthetic data generation.

Our primary contribution is that we designed and implemented the first generative modeling system for relational databases that demonstratively synthesizes realistic data.

Thesis Supervisor: Kalyan Veeramachaneni
Title: Principal Research Scientist

Acknowledgments

Many hours of coding, experimentation, and collaboration have gone into this thesis. I would not have been able to complete it all myself.

First, I'd like to thank Kalyan for being the type of adviser I might like to be one day. He always knew when I needed inspiration, when I needed a push, and when I needed some space to run with my own ideas. This balance between letting me explore new solutions and guiding me was the ultimate key to my success.

There were also many others involved with the project, especially for critical portion of running the final experiments. Thanks to Roy for maintaining and troubleshooting the Feature Factory interface, and Nicole for hiring and training our experimental subjects. Also thanks to my sponsor for providing datasets I could play around with, and Arash for his amazing graphics.

Finally, I'd like acknowledge my friends for all the great memories during my time at MIT: Stephanie, Josh, Denis, Nikki, Anthony, Bennett, Sylvan, Max, and everyone on Burton Third. And of course, Mom, Dad, and Ru for their unwavering love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Businesses are looking to make data-driven decisions by using machine learning methods. Unfortunately, many organizations who wish to adopt these techniques face barriers. Some do not have the resources to collect large datasets that are relevant to their business. Others struggle with hiring data scientists, and have difficulties sharing sensitive data with them.

We contend that such businesses would benefit greatly from the ability to create *synthetic data*, data that is not original but maintains the same mathematical properties and relations. This would result in the ability to generate bulk data on-demand, and publish it freely.

To this end, we introduce the Synthetic Data Vault (SDV), a system that generates synthetic data by building a fully generative model of the original database. The SDV can synthesize data according to the organization's specifications across any complex, relational dataset. In this chapter, we first motivate the thesis by providing some compelling reasons for synthesizing data. We illustrate how the SDV can be applied in industry, and end by providing goals for the thesis.

1.1 Motivations for Synthesizing Data

Our motivation for synthesizing data comes from industry, where organizations are using data to solve predictive problems essential for business. We have identified two

areas where organizations can benefit from synthetic data generation.

1.1.1 Populating Sample Databases

For new start ups, a barrier to using data is not having enough of it. New and traditional machine learning assume a large number of data points that would come with a large userbase. For example, the recently published AlphaGo system samples 30 million data points after analyzing millions of games [13], and ImageNet uses a neural network trained with 15 million images from a publicly available dataset [6]. While businesses may not want to perform sophisticated analysis, the general trend in machine learning is to use more data.

A generative model of a small database is useful for such organizations because they can use the model to create synthetic data in bulk. Companies can use the synthetic data for:

- Performance testing with large amounts of data. With a synthetic data model, they can sample as many datapoints as they want, scaling the data size to many Terabyes, and then evaluate the performance of their algorithm on it.

- Testing software internally. When developing software and debugging, developers who wish to have a sample dataset on their local machine or in their workflow can have synthetic data instead of real data. Furthermore, the synthetic data integrates well with existing applications because it follows the same format as the original.

- Releasing portions of data for marketing outreach. For example, when companies want to share an open source software and demonstrate it on data, they can release the synthetic data instead of real data. The synthetic data follows the same mathematical properties as the original, so its analysis remains valid for real data.

1.1.2 Scaling Data Science Efforts

Organizations wishing to scale their data science efforts must increase the number of people who can work with their data. For example, consider an organization trying to hire freelance data scientists or put data up for a competition on Kaggle [5]. To be able to share data, the organization would need to anonymize sensitive information or remove portions of it entirely. Below, we briefly describe how both these tasks are non-trivial and subject to flaws.

Anonymization

Anonymizing person-specific data is an option that allows organizations to publish data without leaking sensitive information like names or social security numbers. However, deciding which information to anonymize and which to share is a non-trivial task. For example, organizations in the past have freely released the date of birth, gender, and zip code of their customers. Alarmingly, these three pieces of information uniquely identify at least 87% of US citizens [14]. Furthermore, it may be possible to cross-reference information from multiple sources to de-anonymize additional information.

Omission

Omitting sensitive data is a different option that fully protects the privacy. In particular, a k-anonymity scheme purposefully omits individual entries to ensure that any row of data is indistinguishable from at least $k-1$ others [15]. While this is secure, it fundamentally changes the structure of the data. The modifications force anyone working with the data to change their approach.

Only synthetic data can mimic the properties of the original data while also ensuring that the real data is not leaked. By generating synthetic data, the organization does not have to spend resources deciding which data to share, and how to anonymize it.

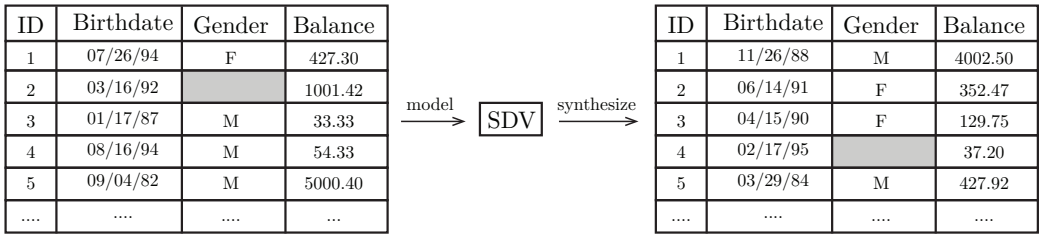

Figure 1-1: Original data that a company has, and the synthetic data that the company produces using the SDV. The synthetic data has the same mathematical properties as the original. It has the same ranges, distribution of values, and covariances. The data is presented in the same format, as in datetime column, `Birthdate`. It even models anomalies, such as some of the genders being missing in the original data.

1.2 Synthetic Data Vault

The Synthetic Data Vault (SDV) represents our solution for data science. It is an end-to-end system that models and synthesizes data automatically such that the synthetic data is virtually indistinguishable from the original. Figure 1-1 illustrates a data sample before it is modeled, and synthetic data that results as an output.

The organization can then store the original customer information internally, and only allow employees with special clearance to view this sensitive data. The synthetic data, however, can be widely spread. The company can share it on competitions such as Kaggle, use it to run predictive analytics, or display it on their website to advertise its services.

1.2.1 Goals

On a high level, we wish to produce an end-to-end system that automates the modeling and synthetic data generation for relational databases. Our success should be measured along three dimensions:

1. **Generalizability**: The SDV should be able to work on all relational databases without any modifications. It should automate as much of the modeling and synthesizing as possible.

2. **Usability**: The SDV should expose a simple API that allows users to specify an input and then perform synthesis to their specification. It should be able to synthesize data for individual cells, columns, tables, or the entire database.

3. **Accuracy**: The SDV should synthesize output that can realistically replace the original data. This requires us to formulate metrics to measure differences between the synthesized data and the original data.

Note the focus on accuracy means that we are not concerned with optimizing computational time or memory. We believe that saving our computations on disk is a reasonable one-time cost for this first iteration of the SDV. Furthermore, our generalizability goal does not include modeling natural language or time series data. These are interesting extensions for future work, but out of scope for the current project.

1.3 Thesis Roadmap

The rest of the thesis is organized as follows:

Chapter 2 provides an overview of the SDV, describing a 4-stage pipeline for a user synthesizing data. Chapter 3 then reviews terminology used for both databases and statistical analysis. We also provide our perspective on the intersection of these two fields.

Chapter 4 covers the technical details behind the SDV's generative modeling method, providing pseudocode to illustrate how we apply the method for an entire database. Chapter 5 then describes the method to synthesize data using the model.

Chapter 6 validates the SDV by using it to model real-world complex datasets, and designing an experiment to measure the effect of working with synthesized data instead of real data. Chapter 7 summarizes our key findings and contributions.

By the end of this thesis, you will understand a new algorithm for modeling relations between tables, and new techniques for data synthesis across multiple tables.

Chapter 2

Overview

This chapter provides an overview of the completed SDV workflow from a user's perspective. The workflow is broken down into four steps, as illustrated in Figure 2-1. First, the user must collect and format the data into a database that the SDV can understand. Then, they provide some basic information about the structure of the database so that SDV knows what to model. Once the SDV finishes its computations, the user is exposed to an API that can perform inference and synthesize data at varying granularities. These steps are the same for all databases.

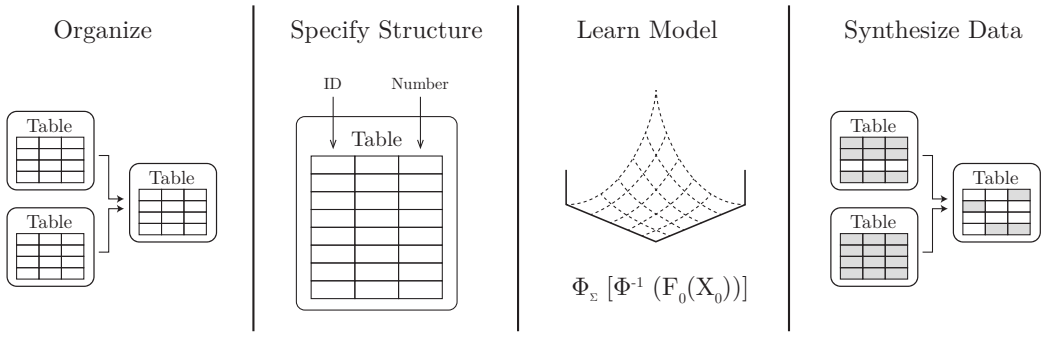

Figure 2-1: The SDV workflow: The user collects and formats the data, specifies the structure and data types, runs the modeling system, and then uses the learned model to synthesize new data.

2.1 Organize

Before supplying the data to the SDV, the user must format the database's data into separate files, one for each table. The SDV makes two assumptions about the relationships between the tables.[1]

- The database should only have many-to-one relationships. While certain databases allow many-to-many relations, these can easily be converted into two many-to-one relations by introducing a new, intermediate table.
- The database should not have circular references. That is, if table A relates to B and B relates to C, then C cannot relate back to A. In practice, we find that if such references are necessary, the layout of the database is failing to properly capture relations in the data on its own. Thus, it is unreasonable to expect a model to understand the complexities behind the relations as well.

2.2 Specify Structure

The user must specify basic information about the structure of each table, and provide it as *metadata* for the database. This specification is similar to a schema in an SQL database.

For each column of data, the user must specify the type of data that is included in the column. The SDV expects that data falls into one of five categories:

1. Number: A numerical value, either an integer or decimal.
2. Categorical: Discrete categories. These can be represented as text or numbers.
3. Datetime: Time information, with a specified format.
4. ID: Either identifying information for the table rows, or references to other tables in the database. These could be represented as numerical values or as text.
5. Text: Raw text that should not be modeled. If there is any structure to the text, the user can provide a regex describing it.

[1]Relationships and other database terminology is described in further detail in the next chapter.

Columns with ID information are special because they contain relationship information between multiple tables. If the ID column of a table references an ID column of another table, the user must specify that table.

Although our goal is to automate much of the modeling framework, we find that asking the user to supply a meta file is unavoidable. In particular, we could not find a way to successfully recover the relations from different tables, because each organization uses different naming conventions for the columns. Furthermore, while it may be possible to automate discovering the type of data, we still run into problems such as discovering the format for datetime columns or differentiating between numbers, categories, and ID. These may be the goals for a future project, which can then be layered on top of the SDV's system.

Furthermore, all SQL databases require a schema, so the information should be readily available to the database administrator. In order to compute the generative model, the SDV accepts a meta file containing all of this information as input.

2.3 Learn Model

The user then invokes the SDV's script to learn the generative model. The SDV iterates through tables sequentially, using a modeling algorithm designed to account for the relations between the tables.

For each table, the SDV discovers a structure of dependence. If other tables reference the current one, dependence exists, and the SDV computes aggregate statistics for the other tables. The aggregate statistics then get added to the original table, forming an extended table. This extended table is then modeled. It captures the generating information for the original table columns, as well as all the dependencies between tables.

The SDV uses some simple optimizations to improve efficiency. It saves all the extended tables and model information to external files, so that subsequent invocations for the same database do not unnecessarily perform the same computations.

2.4 Synthesize Data

After instantiating the SDV for a database, the user is exposed to a simple API with three main functions:

1. `database.get_table`
2. `table.synth_row`
3. `table.synth_children`

The first returns a model for a particular table in the database. Once the table has been found, the user can perform the other two functions using it.

The `synth_row` function both synthesizes rows and infers missing data. If the function is called without any arguments, it synthesizes a complete row. Optionally, the user can specify particular values for any subset of columns. When this happens, the `synth_row` function performs inference and returns the complete row with the missing values filled in.

The `synth_children` function synthesizes complete tables that reference the current table. By applying this function iteratively on the newly-synthesized tables, the user can synthesize an entire database.

The results of both `synth_row` and `synth_children` match the original data exactly. The SDV takes steps to delete extended data, round values, and generate random text for textual columns. The result leads to rows and tables containing fully synthesized data that can be used in place of the original.

Chapter 3

Concepts and Terminology

The goal of building generative models for relational databases involves bringing together work from databases and statistical analysis. For readers who are unfamiliar with either of these areas, this chapter reviews concepts and terminology. The last section offers our perspective on the overlap between the two fields by drawing connections between the concepts.

3.1 Databases

Databases store large amounts of information in the form of tables. A single table represents a particular set of objects, such as pets, web access logs, or stores. It is arranged such that every column represents an attribute of the object (name, age, timestamp, etc.), which means that data in a column is of the same type. Every row represents an instance of the object. In order to distinguish particular instances, there is usually a column that acts as a reference ID. This column is known as the **primary key** of the table; each instance has a unique primary key ID.

In a relational database, there are multiple tables representing different collections of objects. The database is relational because the objects may be connected to each other. For example, many pets could belong to the same store. In order to express this relation, some have columns containing a primary key ID of *another* table. In this example, the pets could have a column that refers to the store ID. Such a column

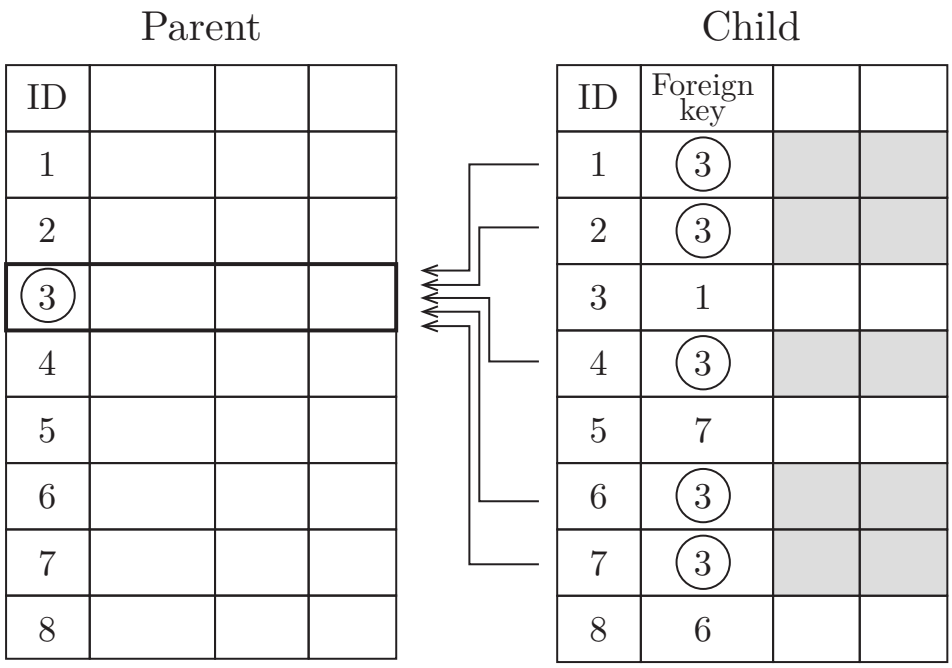

Figure 3-1: Parent and child rows based on key references. The parent's first column contains its primary keys, one of which is "3". The child's second column contains references to the parent. The ones that have a "3" are referring to the same parent row.

is called a **foreign key**. Foreign keys in a table don't have to be unique; many pets could refer to the same store. The mapping is inherently a many-to-one relationship.[1]

In a table relation, the table with the primary key is the **parent table**, while the table with the foreign key references is the **child table**. The names capture the notion that multiple rows in a child table reference the same row in the parent table. Figure 3-1 summarizes this information. The process of finding all the child rows that reference a parent is called a **conditional primary key lookup**.

Note that the relation defines which parent is the table and which is the child. A table may be a parent in one relation, and a child in another relation. In this thesis, we call a table a **leaf table** if it is never a parent in any of its relations.

[1]If the foreign keys are also unique, this becomes a one-to-one relationship. Some database models also allow many-to-many relationships by allowing a foreign key to reference another foreign key, but the SDV assumes this is not the case.

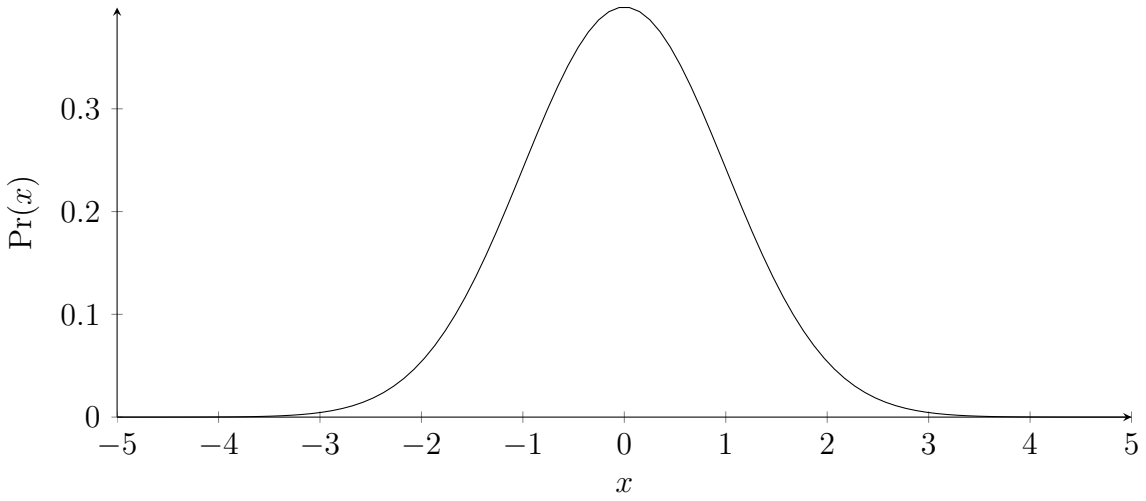

Figure 3-2: The probability frequencies of a standard normal distribution. The probability of observing value x in a Gaussian distribution is $\Pr(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$. The standard normal is a special case where mean $\mu = 0$ and variance $\sigma^2 = 1$.

3.2 Statistics

Statistics involves lists of numerical values that represent different measurements of the same phenomenon, such as a person's age, or height. This measurement is known as a **random variable**, because it can take on different values. However, some values may be more or less frequent than others. The **distribution** of the random variable assigns a probability to each possible value of a measurement trial.

It is possible to encode the probability distribution of a random variable using a **cumulative distribution function (cdf)**. This function accepts an input x that describes a single measurement trial, and returns u, the percentile rank of the measurement in the overall distribution. Mathematically, $u \in [0, 1]$ such that $Pr(measurement \leq x) = u$. If the distribution is d, the *cdf* is $F_d(x) = u$ and its inverse is $F_d^{-1}(u) = x$.

In statistics, there is a special distribution called the **Gaussian** distribution that is fully described by the mean, μ, and variance, σ^2. When $\mu = 0$ and $\sigma^2 = 1$, the Gaussian distribution is called a **standard normal** distribution. Figure 3-2 shows a plot of this distribution. A Gaussian distribution's *cdf* is denoted by $\Phi_{\mu,\sigma^2}(x)$, while a standard normal distribution's *cdf* is simply $\Phi(x)$.

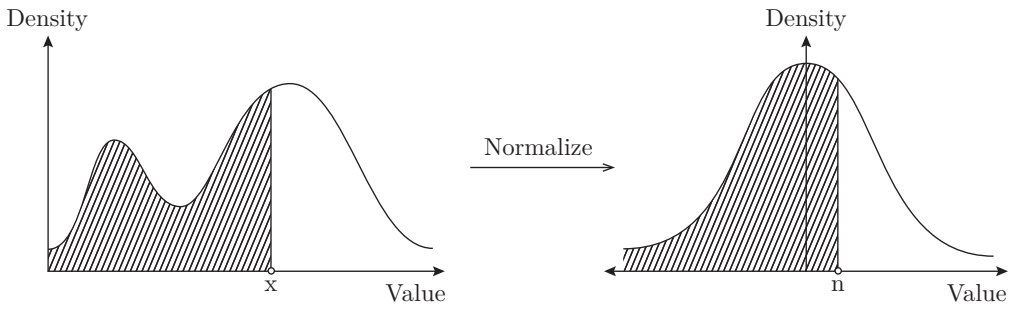

Figure 3-3: A visual depiction of applying the Gaussian Copula process to normalize an observation by applying $n = \Phi^{-1}(F(x))$. Calculating $F(x)$ yields a value $u \in [0, 1]$ representing the proportion of shaded area at the left. Then $\Phi^{-1}(u)$ yields a value n by matching the shaded area in a Gaussian distribution.

Sometimes, a distribution of a random variable may be in a complex or undesirable shape. It is possible to convert the distribution into the shape of another distribution by applying a mathematical transform. The approach we use in this thesis is the **Gaussian Copula** process, which transforms a value x by applying $n = \Phi^{-1}(F_d(x))$. The result of applying the Gaussian Copula process is a new distribution that looks like the standard normal distribution. Figure 3-3 visually depicts this process.

Most real-world problems contain multiple random variables, each with a distribution. The covariance between two distributions measures how values in one distribution are associated with values in another. If a and b are distributions, their covariance is denoted by $\sigma_{ab} = \sigma_{ba}$. A positive covariance means higher values in a tend to yield higher values in b; a negative means the opposite.

When there are n distributions, there exist n^2 ways to calculate the covariance between two distributions.[2] When each of the covariances are placed in a $n \times n$ matrix, the result is a **covariance matrix** denoted by Σ.

[2]The covariance between a distribution and itself is just the variance of the distribution, σ^2.

3.3 Connections

Creating a generative model for a database involves performing statistical analysis. In order to do this, we drew some high level connections between database and statistical concepts as summarized by 3.1. The rest of this section elucidates these connections.

Database Concept	Statistical Concept
Column Data	Random Variables
Table Data	Multivariate Distributions
Relations	Distribution of Distributions

Table 3.1: High-level connecting ideas between the database domain and the statistical domain.

Column Data as a Random Variable

A table column can correspond to a particular attribute or measurement of an object, such as height or weight. The values in the column then form a random variable, which allows us to apply statistical analysis to the column, like calculating the mean μ, or the *cdf* function, F.

Columns with numbers, datetime, or categorical information can be formulated as a random variables: Numbers are essentially the same as measurements. Datetime information can be converted to a number by expressing it as the number of seconds elapsed since Epoch. Categorical information can be converted to numbers by applying a variety of techniques. Other columns that represent text or ID have no corresponding conversions, and therefore no corresponding analogue to random variables.

Table Data as a Multivariate Distribution

The entire table can contain many columns that represent random variables. If this is the case, this data in each row can be condensed to form a single data point in multidimensional space. The entire table then becomes a collection of such multidimensional

points, which is similar to a multivariate random variable.

Multivariate random variables have corresponding multivariate distributions. There also exists a corresponding multivariate Gaussian Copula for them.

Relations as a Distribution of Distributions

In a many-to-one relationship, every row in a parent table is referenced by a subset of rows in the child table. Each subset of rows is a distribution that can be described by statistics such as μ.

If there are n rows in the parent table, then there are n different subsets of children, and n corresponding statistics. We observe that these n statistics can form their own distribution: $d = [\mu_1, \mu_2, \ldots, \mu_n]$.

This new distribution, d, contains numerical values that represent other distributions. Essentially, it is a distribution of distributions. The SDV especially makes use of this idea to model the dependencies in data that are induced by foreign key relations.

Chapter 4

Generative Modeling Method

This chapter covers the technical details of the SDV's modeling phase in the overall workflow presented in Figure 2-1. The goal of the generative modeling phase is to build a complete model for the entire relational database given meta files and tables. Ultimately, the SDV's database modeling method builds generative models for individual tables. However, it performs extra computations to account for the the relations between them using a method called Conditional Parameter Aggregation (CPA). A high-level overview is provided by Figure 4-1.

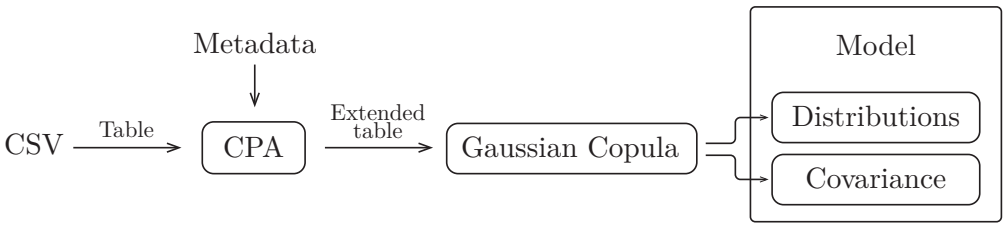

Figure 4-1: An overview of the generative modeling process. Conditional Parameter Aggregation accounts for the foreign key relations across multiple tables. The Gaussian Copula process calculates the overall table model.

This chapter is broken into five sections: Section 4.1 reviews a multivariate generative modeling method we use for a table. This corresponds to the Gaussian Copula and model steps in Figure 4-1, and provides a foundation for our work. Section 4.2 describes extending the generative model to encompass multiple tables. This is called

condition parameter aggregation CPA. The next two sections provides additional adjustments necessary to make the algorithms more generalizable. Finally, Section 4.5 provides the overall logic for applying our technique. This means recursively applying CPA for all tables, in order to model the entire database.

4.1 Standalone Table Model

We define a standalone table as a set of rows and columns that we wish to model independently of any other data. The generative model for a standalone table encompasses all columns that represent numerical data,[1] and it consists of:

- Distributions: The probability distributions of the values in each column
- Covariances: How the value of a column affects the value of another column in the same row

The distribution describes the values in a column, and the covariance describes their dependence. Together, they form a descriptive model of the entire table.

4.1.1 Distribution

A generative model relies on knowing the distribution shapes of each of its columns. The shape of the distribution is described by the *cdf* function, F, but may be expensive to calculate. A simplistic estimate is to assume the original distribution is Gaussian, so that each F is completely defined by a μ and σ^2 value. However, this is not always the case. Instead, we turn to some other common distributions shapes that are parametrized by different values:

- Truncated Gaussian Distribution: Parametrized by the mean μ, variance σ^2, min, and max values
- Uniform Distribution: Parametrized by the min and max values
- Beta Distribution: Parametrized by α and β

[1] Later, we discuss how to convert other types of data, such as datetime or categorical, into numercial data

- Exponential Distribution: Parametrized by the decay λ

If the column's data is not Gaussian, it may be better to use a different distribution. In order to test for this fit, we use the Kolmogorov-Smirnov test [8], which returns a p-value representing the likelihood that the data matches a particular distribution. The distribution with the higher p-value is the distribution we use to determine the *cdf* function. Currently, we decide between truncated Gaussian and uniform distributions, but we provide support to add other distributions.

Note that parameters represent different statistics for each distribution. For this reason, the SDV also keeps track of the type of distribution that was used to model each column. This lets the SDV know how to interpret the parameters at a later stage. For example, if the distribution is uniform, then the parameters represent the min and max, but if it's Beta, then they represent α and β.

4.1.2 Covariance

In addition to the distributions, a generative model must also calculate the covariances between the columns. However, the shape of the distributions might unnecessarily influence the covariance estimates [12].

For this reason, we turn to the multivariate version of the Gaussian Copula described in Section 3.2. The Gaussian Copula removes any bias that the distribution shape may induce, by converting all column distributions to standard normal before finding the covariances. Steps to model a Gaussian Copula are:

1. We are given the columns of the table $0, 1, \ldots, n$, and their respective cumulative distribution functions $F_0, \ldots, F_n$.
2. Go through the table row-by-row. Consider each row as a vector $X = (x_0, x_1, \ldots, x_n)$.
3. Convert the row using the Gaussian Copula:
   $Y = [\Phi^{-1}(F_0(x_0)), \Phi^{-1}(F_1(x_1)), \ldots, \Phi^{-1}(F_n(x_n))]$
   where $\Phi^{-1}(F_i(x_i))$ is the inverse *cdf* of the Gaussian distribution applied to the *cdf* of the original distribution.

4. After all the rows are converted, compute the covariance matrix, Σ of the transformed values in the table.

Together, the parameters for each column distribution, and the covariance matrix Σ becomes the generative model for that table. This model contains all the information from the original table in a compact way, and can be used to synthesize new data for this table.

4.2 Relational Table Model

In a relational database, a table may not be standalone if there are other tables in the database that refer to it. Thus, to fully account for the additional influence a table may have on others, its generative model must encompass information from its child tables. To do this, we developed a method called Conditional Parameter Aggregation (CPA) that specifies how its children's information must be incorporated into the table. Figure 4-2 shows the relevant stage of the pipeline.

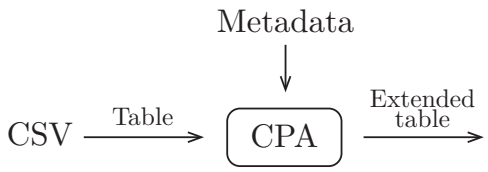

Figure 4-2: Aggregating data from multiple child tables creates an extended table that accounts for the original relations.

This section explains the CPA method. CPA is only necessary when the table being processed is not a leaf table. This means there is at least one other table with a column that references rows in the current one. CPA comprises of 4 steps:

1. Iterate through each row in the table.

2. Perform a conditional primary key lookup in the entire database using the ID of that row. If there are m different foreign key columns that refer to the

current table, then the lookup will yield m sets of rows. We call each set conditional data. Figure 4-3 illustrates such a lookup that identifies $m = 3$ sets of conditional data.

3. For each set of conditional data, perform the Gaussian Copula process. This will yield m sets of distributions, and m sets of covariance matrices, Σ. We call these values *conditional parameters*, because they represent parameters of a model for a subset of data from a child, given a parent ID. This is also shown by Figure 4-3.

4. Place the conditional parameters as additional values for the row in the original table.[2] The new columns are called *derived columns*, shown in Figure 4-4.

5. Add a new derived column that expresses the total number of children for each parent.

The extended table contains both the *original* and *derived* columns. It holds the generating information for the children of each row, so it is essentially a table containing original values and the generative models for its children. The SDV writes a the extended table as a separate CSV file, so we do not have to recalculate CPA for subsequent invocations of the same database.

Subsequently, we can use Gaussian Copula process to create a generative model of the extended table. This model not only captures the covariances between the original columns, but the dependence of the conditional parameters on the values in the original columns. For example, it includes the covariance between original column T_0 and derived column μ_5^2.

4.3 Pre-Processing

Both Gaussian Copula and CPA assume there are no missing entries in the column, and that the values are numerical. When either of assumptions is false, a

[2]Some values repeat because $\Sigma = \Sigma^T$ We drop the repeats to save space.

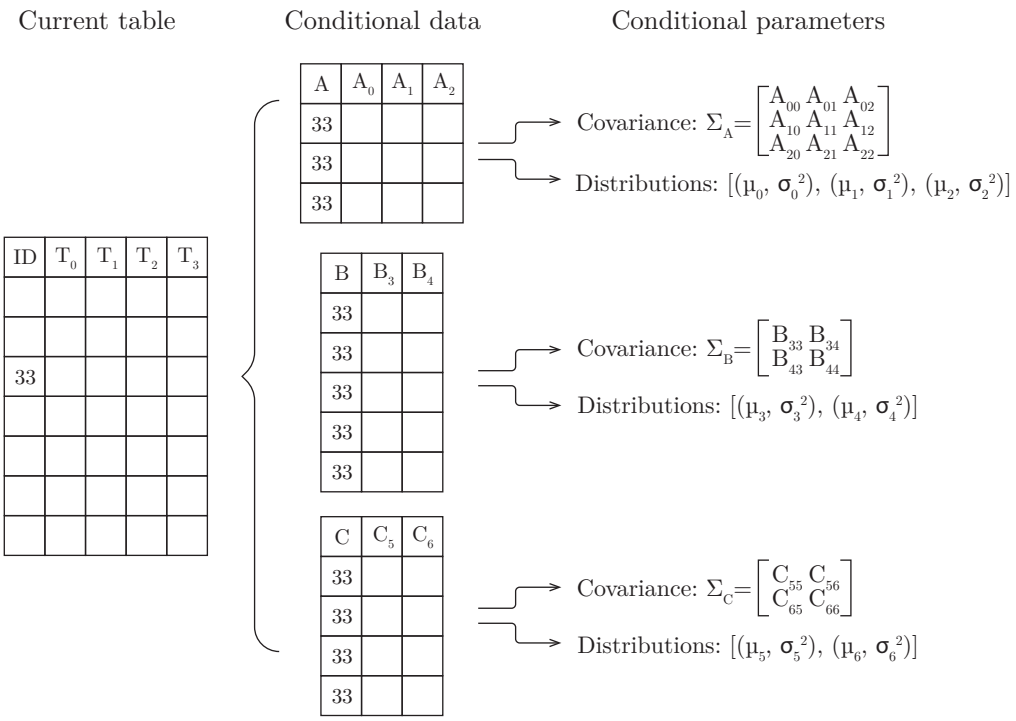

Figure 4-3: An illustration of CPA for a row in table T with primary key "33". Tables A, B, and C refer to table T, so the lookup yields 3 sets of conditional data. Each is modeled using the Gaussian Copula, yielding conditional parameters.

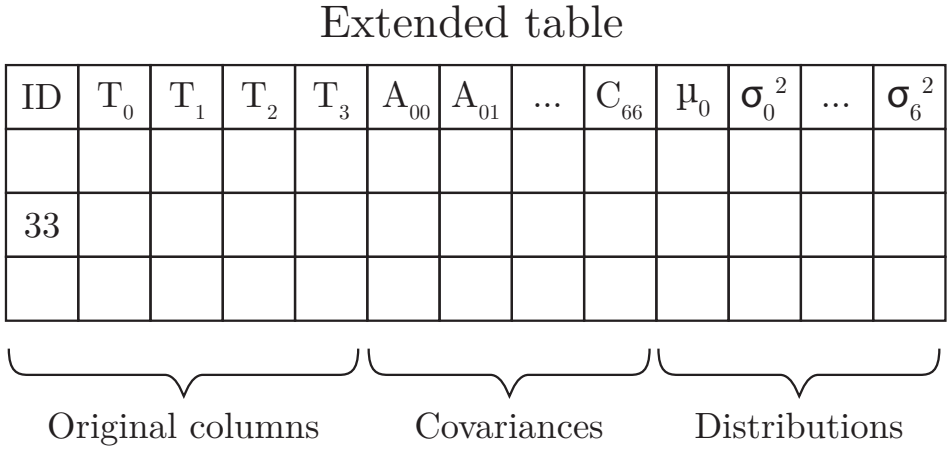

Figure 4-4: The result of CPA. Every lookup for a row yields a value, such as μ_5 or B_{43}. The values form their own columns, resulting in an extended table.

Original Column Type	Replaced Column(s) Type
Categorical	Number
Datetime	Number
Number w/Missing Values	Number & Categorical
Categorical w/Missing Values	Categorical & Categorical
Datetime w/Missing Values	Datetime & Categorical

Table 4.1: Conversions that must be made when pre-processing. If multiple data types are listed, it means that multiple columns are created from the original column.

pre-processing step is invoked. This step ultimately converts a column of one data type into one or more columns of another data type, as summarized by Table 4.1.

Note that some data types might require multiple rounds of pre-processing. For example, a column that is a datetime with missing values is first converted into two columns of type categorical and datetime. Then, those resulting categorical and datetime columns are further converted into number columns.

4.3.1 Missing Values

Missing values in a column cannot simply be ignored because the reasons for which they are missing may reveal some extra information about the data. As an example, consider a table representing people with a column called `weight`, which is missing for some rows. The reasons for missing data falls into one of three categories, so identified by the statistical analysis community [7]:

1. Missing not at random: The data is missing because of what it's supposed to be. Perhaps people who are overweight chose not to disclose their weight, so knowing that the cell is missing probably means the weight is high.

2. Missing at random:[3] The fact that the item is missing is linked with some other piece of data in that row. For example, perhaps a majority of females did not disclose their weight. So knowing that a person is female makes it more likely that the weight column will be missing.

[3]We realize that this is a confusing name. Think of *missing at random* to mean that a random subgroup decided not to supply data.

3. Missing completely at random: The fact that the item is missing tells us nothing about the structure of the rest of the data. For example, the database admin accidentally deleted some of the weights, randomly (oops).

In the first 2 cases, knowing that the value is missing provides further information about the data itself. Therefore, it is important to model missing values overall. Furthermore, a high level goal of the SDV is to model and synthesize data that mimics the format of the original. If the original data has some missing values, the synthesized must too. Modeling the null values solves this problem.

In the final case, it is not imperative that the missing values are considered from a numerical perspective, but the SDV does not know this may be the case. Hence, even though the third case is missing completely at random, the SDV must make a model.

When the SDV encounters any column that has at least 1 missing value, it replaces the column with two columns:

- A column of the same type, with missing values filled-in by randomly choosing non-missing values in the same column.
- A categorical column that contains "Yes" if the original data was present, and "No" if the data was missing for that row.

This solution ensures that the original column contains values for all rows, but also accounts for the fact that some were originally missing.

4.3.2 Categorical

Categorical columns may exist originally in the table, or may be a result pre-processing missing values. Categorical data also cannot be modeled by the Gaussian Copula or CPA.

When it encounters a categorical column, the SDV replaces it with a numerical column containing values in the range $[0, 1]$. To do this, it uses the following method:

1. Sort the categories from most frequently occurring to least.

Categorical

x	Pr (x)
"Yes"	0.4
"No"	0.5
"Maybe"	0.1

Numerical

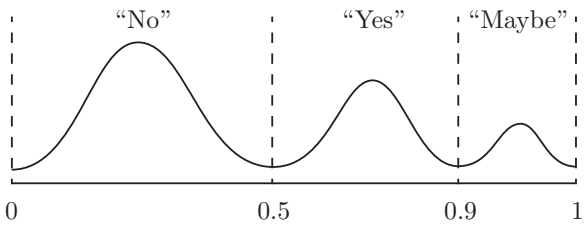

Figure 4-5: The method that converts categorical variables to numerical data. Based on the proportions, "No" is assigned the interval $[0, 0.5]$; "Yes" is assigned $[0.5, 0.9]$, and "Maybe" is assigned $[0.9, 1]$. Each occupies its allocated interval with a Gaussian distribution.

2. Split the interval $[0, 1]$ into sections based on the cumulative probability for each category.
3. To convert a category, find the interval $[a, b] \in [0, 1]$ that corresponds to the category.
4. Chose value between a and b by sampling from a truncated Gaussian distribution with μ at the center of the interval, and $\sigma = \frac{b-a}{6}$.

Figure 4-5 shows a visual depiction of this conversion.

Note that while Gaussian distributions are completely defined by μ and σ^2 , the same is not true for these categorical distributions. Instead, they require new parameters representing the proportions of each of the c categories, $p_0, p_1, \ldots, p_c$ with $0 \leq p_i \leq 1$ and $\sum_i p_i = 1$. These are the conditional parameters that are put in the extended table for categorical columns.[4]

Choosing a value using a Gaussian distribution gives dense areas at the center of each interval, but ensures that the numbers are essentially different. The inverse is also easy to compute: Given a value $v \in [0, 1]$, we find the interval that v corresponds to and return its category.

[4]We save $p_0 \ldots p_{i-1}$ because the last proportion p_i can be calculated from the others.

4.3.3 Datetime

Finally, many tables contain information containing times or dates that is represented as text. The SDV replaces such columns with numerical values. This is relatively straightforward, as any timestamp can be expressed as the number of seconds past Epoch (January 1, 1970). If timestamp represents a time before Epoch, then the numerical value is negative (number of seconds until Epoch).

4.4 Nomenclature

Both pre-processing and CPA add new columns to the table. A standardized nomenclature for these new columns is necessary for two reasons:

- No namespace guarantees: From the CPA discussion in Figure 4-3, recall the figure displayed child rows in organized sets, A, B, and C, and numbered each column uniquely from A_0 to A_6. In practice, sets and column names are not guaranteed to be unique, and a single parent table may even have multiple foreign keys coming from the same table.

- Usability: Generic names like A_{00} and μ_1 provide no explanation about how the columns were derived from the conditional key lookups, and may be confusing to the users.

Thus, our goals were to ensure that every new column has a unique name, and that every unique name clearly defines the computation that created it.

4.4.1 Definition

The final nomenclature for column names is presented below:

$$
\begin{aligned}
\texttt{<name>} &= \texttt{<derived\_name>|<original\_name>} \\
\texttt{<derived\_name>} &= \texttt{<missing\_values>|<CPA>} \\
\texttt{<missing\_values>} &= \texttt{?<name>} \\
\texttt{<CPA>} &= \texttt{<child\_table>:<foreign\_key>(<value>)} \\
\texttt{<value>} &= \text{“count”}\texttt{|<name>*<name>|<name>@int|<name>@category}
\end{aligned}
$$

Original column names are those that have been defined and inputted by the creator of the database. The SDV cannot control what these may be.

Derived columns can either be the result of having an original column with missing values, or the result of CPA. For CPA, the prefix is based on the name of the child table and foreign key. These two pieces of information precisely define the conditional primary key lookup that was performed before aggregating the data. The type of aggregation is then described by the value inside the parenthesis. This can be one of four aggregations:

1. “count”: The columns represents the number of child rows that were returned by the conditional primary key lookup. All values must be integers.
2. `<name>*<name>`: This column represents the covariance between two columns.
3. `<name>@int`: This column represents a distribution parameter for the column. The particular quantity this describes depends on type of distribution. This is not valid for categorical columns.
4. `<name>@category`: This column represents the probability that a member of the distribution falls into a specific category. This is only valid for categorical columns. All values must fall in the range $[0, 1]$.

The nomenclature highlights a recursive nature to the database, as each instance of `<name>` is replaced by another column that follows the same nomenclature. The

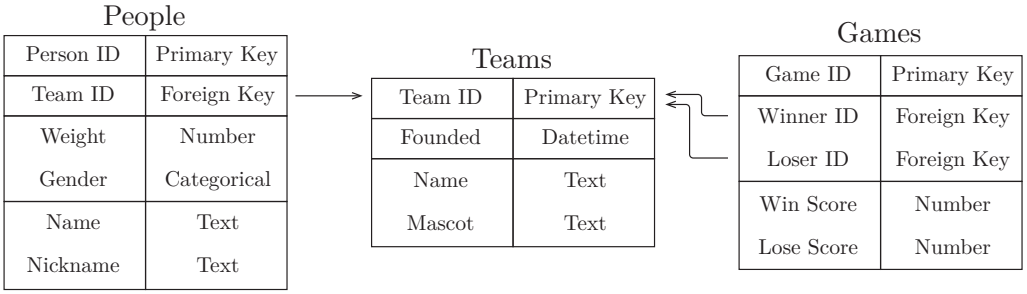

Figure 4-6: The database schema for our example. Arrows are drawn from the foreign key columns to the primary key they reference. Note that the games table has two separate foreign keys that reference the same teams table. Assume that the numerical `weight` column may contain missing cells.

recursion ends with the column names that are provided in the original input data. Simply by viewing the column names, it is possible to reverse-engineer the structure of the entire database.

4.4.2 Illustration

This section illustrates nomenclature on a sample database. Our sample database represents athletes playing games. There are tables representing teams, games, and people. Many people can belong to the same team. A single team can also win or lose many games. Figure 4-6 summarizes this information using a visual representation.

The teams table is a parent table with people and games as its children. Note that people and games do not have any other tables referencing their primary keys (`personID` and `gameID`), making them leaf tables.

When the SDV is given the information for this database, it pulls out all primary key, foreign key, and text columns because they do not represent random variables. The remaining data is involved in the CPA and Gaussian Copula process. The SDV models the people and games table before modeling the teams, because all children of a table must be processed first.

When it is done, the tables may contain some derived columns based on the possible null values and the CPA step. Table 4.2 lists all the modeled columns. Our

nomenclature makes it easy to comprehend the process that was used to calculate the values in each of the columns.

Note that categorical variables with c categories only have $c-1$ categories represented in the derived columns. Therefore, even though the gender of a person may be "Male" or "Female", there is only one column representing the proportions. Similarly, `?weight` is a derived categorical column because the original `weight` column had missing values. The derived column has both "YES" and "NO" entries, but only one of those proportions is necessary to store.

Also note that the number of parameters in the distribution for different numerical fields may vary. For example, `weight` has 4 parameters (`weight@0` through `weight@3`), while `winScore` and `loseScore` only have 2. This happens because the distribution shape of `weight` is different from `winScore` and `loseScore`. In this particular case, `weight` may be a truncated normal distribution that needs 4 parameters for μ, σ^2, min, and max; the others may be uniform distributions that just require 2 parameters for min and max.

Finally, observe that the process of CPA can continue to make new columns if the teams table had a foreign key reference called `leagueID` to parent table that represents leagues (i.e. one league contains many teams). Then the nomenclature would nest the column names of teams as part of CPA. The model would create columns with names such as:

```
teams:leagueID(people:teamID(gender*?weight)*games:winnerID(count))
```

4.5 Database Modeling

This final section describes the overall system by providing control logic for modeling an entire database. This consists of applying CPA recursively to calculate the model of the entire database.

We let D represent a database consisting of many tables, T. The relations between the tables are known, so we let $\mathcal{C}(T)$ represent the set of T's children, and $\mathcal{P}(T)$

	Orig.	**Derived**		
games	winScore loseScore			
people	weight gender	?weight		
teams	founded	**people:teamID**	**dist**	people:teamID(gender@MALE) people:teamID(weight@0) people:teamID(weight@1) people:teamID(weight@2) people:teamID(weight@3) people:teamID(?weight@YES)
			cov	people:teamID(gender*gender) people:teamID(weight*weight) people:teamID(?weight*?weight) people:teamID(gender*weight) people:teamID(gender*?weight) people:teamID(weight*?weight)
			count	people:teamID(count)
		games:winnerID	**dist**	games:winnerID(winScore@0) games:winnerID(winScore@1) games:winnerID(loseScore@0) games:winnerID(loseScore@1)
			cov	games:winnerID(winScore*winScore) games:winnerID(loseScore*loseScore) games:winnerID(winScore*loseScore)
			count	games:winnerID(count)
		games:loserID	**dist**	games:loserID(winScore@0) games:loserID(winScore@1) games:loserID(loseScore@0) games:loserID(loseScore@1)
			cov	games:loserID(winScore*winScore) games:loserID(loseScore*loseScore) games:loserID(winScore*loseScore)
			count	games:loserID(count)

Table 4.2: The original modeled columns and derived columns of each table in the sample database. These are shown in normal text. The bold helps organize the columns by reference (`table_name:foreign_key`) and by aggregation (distribution, covariance, or count).

represent the set of T's parents. Finally, we assume that our logic has access to CPA and pre-processing method we have described. Other mathematical functions include the *cdf* function, F, and the covariance Σ.

The CPA method works across a parent-child relationship. However, the children may have more children, so we must apply the CPA recursively down all of the parent's descendants. We call this recursive approach Recursive Conditional Parameter Aggregation, or RCPA. Algorithm 1 provides the logic for RCPA.

Algorithm 1 A recursive application of CPA to add derived columns to T.

```
1: function RCPA(T)
2:     for all C ∈ 𝒞(T) do
3:         RCPA(C)
4:     T ← CPA(T)
5:     T ← PreProcess(T)
```

Note that the CPA method returns the extended table. Line 4 saves the extended table as T. Finally, line 5 pre-processes T to convert the values into numerical data. The base case of this algorithm is for leaf tables, tables where $\mathcal{C}(T) = \emptyset$. Such tables are guaranteed by our non-circularity constraint.

When the SDV creates the overall model, it applies RCPA and uses the results to calculate the database model. The SDV's modeling algorithm calls the RCPA method on all tables without parents. Because RCPA is recursive, this ensures that all tables in the database ultimately go through the CPA method. Afterwards, it calculates the *cdf* functions, given by F, as well as the covariances by using the Gaussian Copula for all extended tables. The logic is given in Algorithm 2.

Algorithm 2 The overall modeling logic for the SDV for database D.

1: **function** SDV-MODEL(D)
2: **for all** $T \in D$ s.t. $\mathcal{P}(T) = \emptyset$ **do**
3: RCPA(T)
4: $cdf \leftarrow \emptyset$
5: $cov \leftarrow \emptyset$
6: **for all** $T \in D$ **do**
7: $cdf \leftarrow cdf \cup F(T)$
8: $cov \leftarrow cov \cup \Sigma_{(\Phi^{-1}(F(T))}$
9: **return** cdf, cov

The algorithm saves and returns all the *cdf* and covariances of the tables. The *cdf* functions are calculated using the table returned by the extend function. The covariance is calculated after applying the Gaussian Copula to that table. Together, the *cdf* and covariances form the generative model for database D. When this function returns, the user can control the amount and type of data to synthesize.

In summary, the overall database model saves the following for every table:

- The extended table (calculated by Algorithm 1)
- The *cdfs* of columns in the extended table (returned by Algorithm 2)
- The covariances of columns in the extended table (returned by Algorithm 2)

Chapter 5

Data Synthesis

This chapter presents the details of the last step in the SDV's workflow: Synthesizing data based on the calculated database model.

We break up the synthesis into two categories:

- **Model-Based**: The user wishes to synthesize data relying on the model that has been computed. For example, a user may want to synthesize the entire database of their customer information.
- **Knowledge-Based**: The user already has some information about the data, and wishes to synthesize the rest it. For example, the user may want to synthesize information for particular types of customers (female, age 22, etc.).

The SDV can perform both types of synthesis. Each section of this chapter provides details for the two cases. The final section presents our API endpoints.

5.1 Model-Based

Model-based synthesis is based on being able to sample data from the calculated distribution and covariances. The modeling was learned using pre-processed numerical values that represent numbers, datetime, categories, and missing values. Once we sample from the model, we can factor in the primary key and foreign key relations to synthesize tables, and ultimately the entire database.

5.1.1 Sampling Numerical Values

All numerical values can be sampled from the distributions and covariances of the columns. Call the set of *cdf* functions F, and the covariance matrix Σ. The method to sample numerical values is given by algorithm 3. Assume that there are n columns, so that $|\Sigma| = |F| = n$.

Algorithm 3 Sampling numerical values from distribution and covariances of the columns.

1: **function** SAMPLE(F, Σ)
2: $v \leftarrow$ random n-dimensional Guassian vector
3: Find Cholesky decomposition, $LL^T = \Sigma$
4: $u \leftarrow Lv$
5: $x \leftarrow [F_0^{-1}(\Phi(u_0)), F_1^{-1}(\Phi(u_1)), \ldots F_n^{-1}(\Phi(u_n))]$
6: **return** x

Line 4 of this algorithm uncovers a vector, u, in Copula space. Then, line 5 converts it back to the original space by applying the inverse of the Gaussian Copula. The returned vector, x, provides a value for all columns that were converted to numerical data (numbers, categorical, datetime, and missing values).

Once the numerical value is returned, we can post-process it to form data that looks like the original. This is accomplished by:

- Converting back from numerical values to datetime or categorical values
- Removing values for columns that were not originally in the table. This includes all derived columns from CPA.
- Making values blank if they are supposed to be missing by looking at the binary ‘‘`Yes`’’ or ‘‘`No`’’ value that is sampled.

5.1.2 Row Synthesis

Overall row synthesis relies on sampling. We use two separate methods depending on if the row does or does not have any parents.

To synthesize a row with no parents (and therefore, no foreign key references), we use the overall *cdfs* and covariance computed for its table, T_F and T_Σ. To synthesize

a row with a parent, we recall that its parent row, p, has conditional parameters that describe the *cdfs* and covariances for its children, p_F and p_Σ. These are the values we use to generate the child. Both methods are shown in Algorithm 4.

Algorithm 4 Making a row based on information in the table T or in the parent row p.

```
1: function MakeRowFromTable(T)
2:     id ← random unique ID value
3:     x ← Sample(T_F, T_Σ)
4:     return [id, x]
5:
6: function MakeRowFromParent(p)
7:     id ← random unique ID value
8:     foreign key ← ID of p
9:     x ← Sample(p_F, p_Σ)
10:    return [id, foreign key, x]
```

The first function, MakeRowFromTable expects an extended table T as input. This can be either the original extended table, or a synthetic version of the extended table. The second function MakeRowFromParent expects a single row, p, containing all values from the derived columns as input. Similar to the first function, p can be either an original row or a synthesized row. Note that both returned values require post-processing to look like the original version of the data.

5.1.3 Database Synthesis

Synthesizing the entire database just consists of synthesizing multiple rows and child rows recursively. We begin with a table that has no parents, and call the MakeRowFromTable to generate rows for that table. Using the rows from that table, we can create the children. Recall that each parent row, p, also stores the number of children it contains, p_n. We can use this number to call MakeRowFromParent the appropriate number of times. Finally, we recurse to synthesize the children of those children until the entire database is synthesized. The logic is shown by Algorithm 5.

Algorithm 5 The overall database synthesis logic for the SDV.

```
1: function SDV-Synthesize(D)
2:     for all T ∈ D s.t. P(T) = ∅ do
3:         repeat
4:             row ← MakeRowFromTable(T)
5:             MakeChildrenRows(row)
6:         until reached user-defined threshold
7:
8: function MakeChildrenRows(p)
9:     if p has children then
10:        repeat
11:            child ← MakeRowFromParent(p)
12:            MakeChildrenRows(child)
13:        until reached p_n children
```

We envision that a majority of use-cases will be model-based. They will require the user to synthesize the entire database, or a subset of tables in those databases.

5.2 Knowledge-Based

In this section, we briefly describe algorithms we use if the user wants to synthesize data based on prior knowledge they already have. For example, if user is synthesizing data for internally testing an application, they may realize that the application needs a balance of values. As a result, the user may decide to synthesize rows for underrepresented female customers only.

This requires two modifications from the model-based method. First, the sampling method from Algorithm 3 no longer works because some of the values included in F and Σ have already been observed. This requires us to perform a special update to uncover a new F' and Σ' for just the unobserved data. Second, it requires us to infer what the parent might be based on the value that the user provides.

5.2.1 Sampling Updates

If some values are already observed and inputted by the user, then original sampling will not work by itself, because it will return synthesized values for all columns. To

account for observed data, it is necessary to update update the Σ matrix, as well as the mean vector μ. Initially, $\mu = 0$ due to the Gaussian Copula process.

Let k represent all the observed (known) variables, and u represent the unobserved (unknown) variables the user wishes to synthesize. Then we can rearrange the Σ matrix and μ vector to bring all the unknown variables to the top:

$$\Sigma = \begin{bmatrix} \Sigma_{uu} & \Sigma_{uk} \\ \Sigma_{ku} & \Sigma_{kk} \end{bmatrix}$$
$$\mu = \begin{bmatrix} \mu_u \\ \mu_k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

With this configuration, the SDV can update Σ and μ with the known observations to get a new Σ' and μ' for just the unknown.

$$\begin{aligned} \Sigma' &= \Sigma_{uu} - \Sigma_{uk}\Sigma_{kk}^{-1}\Sigma_{ku} \\ \mu' &= \mu_u + \Sigma_{uk}\Sigma_{kk}^{-1}\left(obs - \mu_k\right) \\ &= \Sigma_{uk}\Sigma_{kk}^{-1}obs \end{aligned}$$

Where *obs* is the user-inputted vector containing the known values. Note that the Σ' matrix has dimensions $|u| \times |u|$ and the μ' matrix has exactly $|u|$ elements. This is because they only describe the relations for the columns with unobserved values.

Now, the SDV knows the new Σ' and μ', along with the corresponding *cdf* functions for the unknown variables $F_i, i \in u$. These new values can be used in the sampling algorithm (Algorithm 3) with a slight modification: In step 4, we add the μ to the vector u. This will return all the values in the row that contain numerical information, some of which is post-processed back into categorical or datetime information. However, it does not include foreign key information, which is why we need to perform inference to find the parent.

5.2.2 Parent Inference

If the user has observed certain values for a row and the row has parents, then it is necessary for us to infer what the parent row may be.

Recall that each parent row, p contains conditional parameters that describe the covariances, p_Σ, and *cdfs*, p_F, of its children, so the problem of picking a foreign key simplifies into a log likelihood estimate. For the given data, x, the probability of x belonging to some parent p depends on p_Σ and p_F. This, in turn, is described by the Gaussian Copula:

$$-\log\left(\mathcal{L}_p(x)\right) = -\log\Phi_{p_\Sigma}\left[\Phi^{-1}\left(p_{F_0}\left(x_0\right)\right), \Phi^{-1}\left(p_{F_1}\left(x_1\right)\right), \ldots, \Phi^{-1}\left(p_{F_n}\left(x_n\right)\right)\right]$$

The SDV chooses a parent row of x from a weighted distribution of $-\log\left(\mathcal{L}_p(x)\right)$, $\forall p$. The foreign key of x is the primary key of parent p.

Note that the value $\Phi^{-1}\left(p_{F_i}(x_i)\right) = \pm\infty$ if $p_{F_i}(x_i) = 1$ or 0, making the overall log likelihood approach 0. This happens when the child's data is out of bounds for a parent. For example, if the conditional parameters in p define min and max and the observed row is not in the interval, then p is not a good candidate for a parent.

The overall SDV is able to perform many types of synthesis and inference based on a combination of all the algorithms presented in this section. Given any set of parent or children rows and columns, the SDV can ultimately synthesize the missing values and return them to the user in the same format as the original table.

5.3 API Endpoints

When the SDV is ready for the synthesis stage, it provides the user with a `database` object, from which the user can access individual tables with `database.get_table(name)`. The table object is used for the synthesis.

We have packed both model-based and knowledge-based synthesis in two synthesis endpoints. The first is `table.synth_row`, that allows the user to synthesize a full row

based on the table or its parent rows, while also performing updates based on observed values. The second is `table.synth_children`, that allows the user to generate all children based on a parent table. This method is a convenient packaging of the MakeRowFromParent algorithm that allows the user to easily synthesize full tables and databases.

5.3.1 `table.synth_row`

If they are synthesizing a full row, the user can just call the `synth_row` function without any arguments. This generates all of the modeled data. The SDV generates a unique primary key, as well as any textual data that is not modeled. As a final step, the SDV formats the data to mimic the original. This means performing the following checks and transformations:

1. If the column `<x>` has a corresponding categorical column `?<x>` check its value. If `?<x>` = ``NO'' then the value should be missing. Set the value of `<x>` to null.
2. Remove all columns that were not in the original table.
3. If the original column was a datetime, take the numerical value and convert it back to a datetime with a user-provided time regex.
4. If the original column was a category, perform the transform from Section 4.3.2 in reverse to recover the correct category.

As keyword arguments, the user can input any observed values for columns that exist in the table. The SDV performs the appropriate inference to synthesize a full row based on the input. These can include derived columns too, because derived columns are modeled by the SDV. Table 5.1 shows some examples.

5.3.2 `table.synth_children`

When calling the `synth_children` function, the SDV synthesize entire tables that represent children of the current table. The number of children generated for each unique primary key of the table are based on the value of the derived `count` column.

Command	English Description
`customer.synth_row()`	Synthesize a completely new customer
`customer.synth_row(gender=F)`	Synthesize a female customer
`customer.synth_row(?weight=No)`	Synthesize customer with missing weight

Table 5.1: Example commands using the `synth_row` function to create new stores. Original columns and derived columns can be inputs to the system.

This function completely generates the all the columns of the children table, including any other foreign key constraints that the children may have. This function is intended to help the user generate entirely new databases. The user first calls `synth_row` on every row in the parent table, and the `synth_children` recursively until the entire database is synthesized.

This meets our usability for the SDV: Provide a simple interface for the user that gives them control to synthesize data at any granularity. The cell and row granularities are covered by the `synth_row` method, while the table and database granularities are covered by `synth_children`.

Chapter 6

Experiments

In this chapter, we perform experiments to validate the SDV's ability synthesize realistic data. The experiments in this section evaluate the SDV based on the two motivations described in section 1.1. These are:

1. Populating Sample Databases
2. Scaling Data Science Efforts

For each of these areas, we designed an approach to observe the degree to which the SDV accomplishes these goals. The first section discusses a subjective approach to understanding what the SDV is capable of understanding when populating a sample database. The second section describes a crowdsourced experiment that measured a data scientist's ability to work with anonymized data from the SDV.

6.1 Populating Sample Databases

The first motivation for building the SDV is so that industries can use a generative model to populate sample databases with realistic data. There is generally a difference in data quality between internal datasets from industry, and published datasets for data science competitions. Therefore, it was necessary for us to find a real data source to accurately observe how the SDV performs with the added complexity of a real-world system.

Table	# Rows	# Columns	# Child Tables	# Parent Tables
Employees	1818	206	4	0
Role	361	8	0	1
Promotions	24	28	0	1
Performance	1816	27	0	2
Assessments	9269	130	2	3
Assessment Details	4848	26	0	3
Questions	120	9	3	0
Question Details	10488	37	1	1
Objectives	1907	97	2	1
Objective Details	2193	27	1	1

Table 6.1: Summary statistics for each of the 10 tables in our sponsor's HR dataset.

One of our sponsors, a major software consulting firm, provided us with a dataset that we used to analyze the SDV. This section first describes the dataset in detail, and then the method we used to synthesize the data. Finally, we describe some observations and lessons learned from modeling the dataset.

6.1.1 Dataset

Our sponsor's dataset came from human resources, and described the career goals and reviews for 1818 of their employees. It also contained some information about their quarterly reviews. There were 10 interconnected tables describing this information.

Table 6.1 provides some metadata about the tables, rows, columns, and relations. This dataset contained more relations and interconnected tables than any publicly available relational dataset that we could find.

6.1.2 Method

The ultimate goal was to use the SDV to synthesize data for each of the tables, such that our sponsor could use the synthesized tables for their internal applications. One particular application was used by supervisors to manage their subordinates' status. Our sponsor agreed to plug in the synthesized data as the back-end for this application, and then run the application using real-life scenarios:

1. Querying employee information
2. Querying employee objectives
3. Adding and updating objectives
4. Querying a subordinate's data by a supervisor
5. Adding and updating a subordinate's data by a supervisor

We worked with our sponsor to discover and fix any problems in the synthesized data. After sharing the synthesized data, our sponsors provided us with some feedback from running the test cases. We noted down the problems, and manually fixed them until the tests were able to run fully.

6.1.3 Results

Our sponsor was easily able to import the synthesized data, provided as CSV files, into their database system. The SDV was able to synthesize primary keys and accurately connect rows of different tables using them.

After running the test scenarios, all problems they reported fell into two categories: Hard constraints, and self relations.

Hard Constraints

Hard constraints are logical constraints that hold true for every row or column in a table. An example of this are columns that contain begin and end dates of performance reviews for every user. It is generally assumed that the end date will be after the start date.

From running the synthesized data, our sponsor found that some hard constraints did not hold for about 5% of all rows. The particular constraints they cited were:

- Datetime value comparisons. The period begin date must be before the period end date. The date of the review must be between the begin and end date.
- Missing value based on orderings. Supervisors must first submit their appraisals in a particular order: objectives, assessments, feedback, and then the summary for each subordinate. Thus, if assessment is missing (hasn't been completed),

then the feedback and final summary must be missing too. If the feedback is missing, then the final summary must be missing.

- Exact number of foreign key references. Employees must have exactly 1 performance review per supervisor for the quarter.

The SDV sometimes synthesizes rows that break these constraints due to the probabilistic nature of its synthesizing algorithm. For example, the SDV correctly identifies a positive covariance between a missing feedback cell and a missing final summary cell. However, it treats that covariance as a probabilistically high likelihood of the two values being missing together. When synthesizing many rows, a few of those rows may represent an unlikely scenario.

To automatically fix the issue of hard constraints, it's necessary to perform logical checks on every subset of columns, and considering hard constraints when the check holds for every row. While automatic, this comes with a significant increase in processing time.

If there are only few hard constraints, the user can easily perform rejection sampling on each synthesized row. Because a relatively few percent of synthesized values break the constraints, the total time for synthesizing does not by much.

Self Relations

The second category of mistakes all occurred when the SDV was synthesizing new employees. In addition to an employee ID, there were two additional columns that contained information about the employee's career counselor and direct supervisor. The career counselor and supervisor were also employees, so the employees table had, in effect, a relation to itself. In addition, the supervisor and career counselor had some more non-circularity constraints: If person A was the supervisor of person B, then B could not also be the supervisor of A.

None of these relations were properly captured by the SDV because it only models foreign-key relations as being from a parent table to a different child table. A simple solution to this would be to create a new table whose purpose is to connect the columns of other tables. Note that a self-relation is a type of circular key dependence

that the SDV assumes does not exist in the database. Even from manually inspecting the data, it was difficult to discern the self-relation, as well as the supervisor hierarchy that it represented.

Overall, we found that the SDV correctly synthesized most data successfully for our sponsor's use case of test data creation.

6.2 Scaling Data Science Efforts

In this section, we describe a crowdsourcing experiment designed to validate the SDV's ability to synthesize data for the purposes of anonymization.

The overall goal was to test a data scientist's ability to work with datasets that were synthesized. The ultimate question is if data scientists could work with synthesized data as easily as the original data.

In order to test this, we found publicly available relational datasets with prediction problems for a particular column. For each dataset, we performed the following steps:

1. Run the SDV on the dataset to create the generative model.
2. Use the model to synthesize data with varying degrees of noise.
3. Hire data scientists to solve prediction problem with a particular version of the dataset (synthesized or the original).

This section describes the experiment process. First, we provide details about the datasets, and the method used to synthesize the data. Second, we describe the experimental setup with 4 conditions. Finally, we discuss preliminary results.

6.2.1 Datasets

We found a total of 5 relational datasets to use for the experiment. Two came from an online relational dataset repository [9], and three from came from Kaggle [5]. Table 6.2 provides a summary of each dataset. The prediction problems for each of the datasets was turned into a classification problem by discretizing the target column's values, if they weren't originally categorical.

Dataset Name	Source	# Tables	# Classes
Biodegradability	Relational Repo	5	5
Mutagenesis	Relational Repo	3	2
Airbnb	Kaggle	4	12
Rossmann	Kaggle	2	8
Telstra	Kaggle	5	3

Table 6.2: Summaries of the five relational datasets used for the crowdsourcing experiment. The final column refers to the number of classes that the prediction problem encompasses.

The rest of this section provides schemas and prediction problems for each of the datasets.

Biodegradability

The first dataset describes various chemical compounds in terms of molecules, atoms, and bonds [3]. Figure 6-1 shows the layout of this dataset.

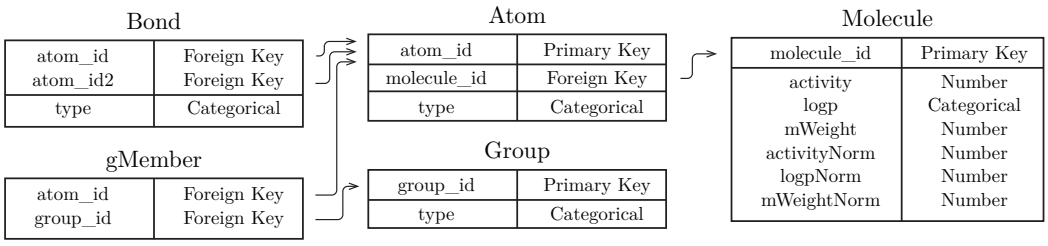

Figure 6-1: The schema for the biodegradability dataset. Molecules consist of multiple atoms. Two atoms are joined by bonds, and multiple atoms can be part of an atom group.

The prediction problem is the biodegradability of each molecule in water, as measured by the column `logp` in the `Molecule` table. The `logp` value describes the half-life of the biodegradation for the molecule. For this experiment the `logp` values were discretized into 5 classes, and the objective was to predict the class that the molecule belongs to.

To create a synthetic database for this prediction problem, the SDV first synthesizes new molecules. From those molecules, it synthesizes new atoms, and from those atoms, it creates new bonds and group members. Note that is it is not necessary to synthesize new groups, because a row in `Group` is not a child of molecule.

Mutagenesis

Similar to the biodegradability dataset, the mutagenesis dataset [4] is also related to chemical compounds described by molecules, atoms, and bonds as shown by Figure 6-2.

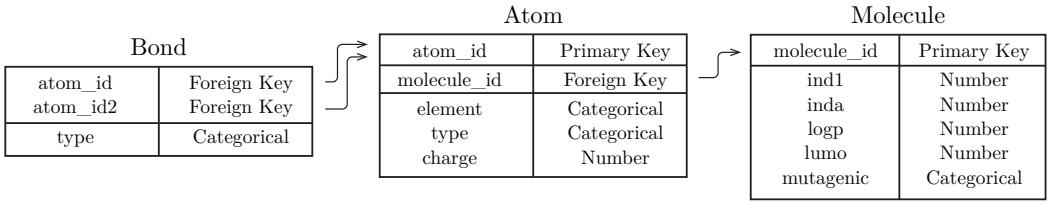

Figure 6-2: The schema for the mutagenesis dataset. The overall structure is the same as for biodegradability, but there is no `gMember` or `Group` tables associated with the dataset.

The objective of this prediction problem was to predict the `mutagenic` column in the `Molecule` table. The term mutagenicity refers to the tendency of a chemical to cause mutations in a strand of DNA. Thus the `mutagenic` column is binary, and contains either a `'yes'` or `'no'` value.

Creating synthetic data was straightforward for the SDV: Create new molecules, new atoms for those molecules, and new bonds for those atoms. Thus, all three tables needed to be synthesized for the prediction problem.

Airbnb

The Airbnb datasets comes from a Kaggle competition [2] hosed by the lodging site Airbnb [1]. It consists of web access log data from each of its users, as described in Figure 6-3.

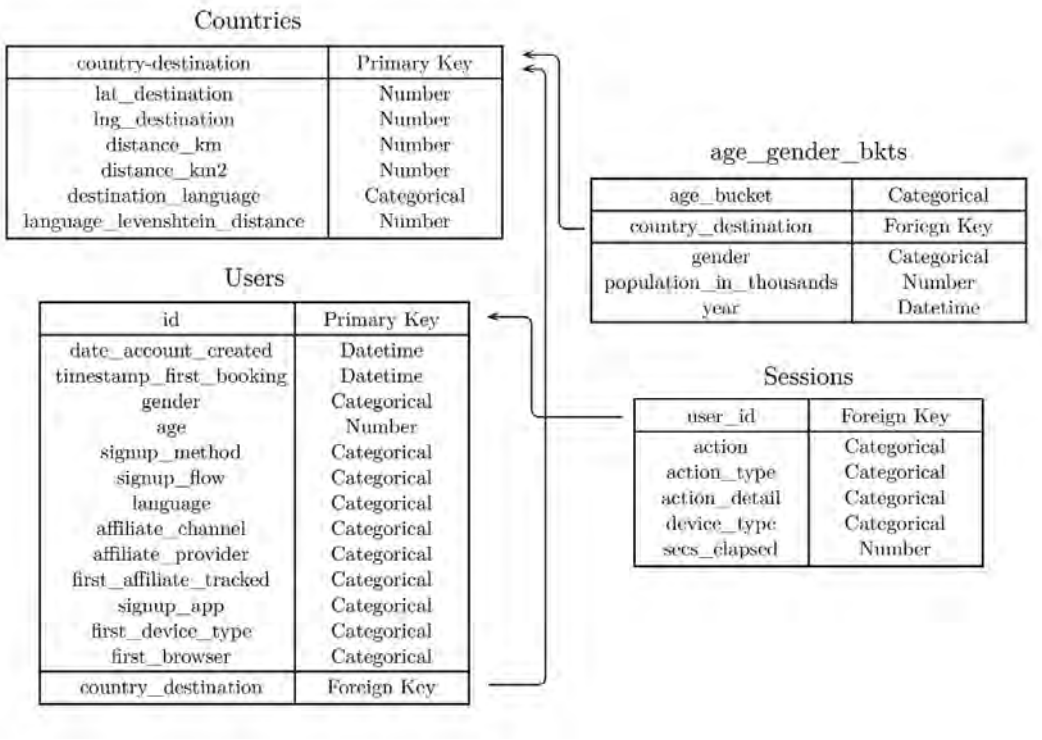

Figure 6-3: The schema for the Airbnb dataset. Each user is an account made on Airbnb, and each session describes a particular access made to the website. The `Countries` table provides general information about the country, while `age_gender_bkt` provides information about people traveling to those countries.

The prediction problem for this dataset is `country_destination` from the `Users` column. This represents the country that a particular user booked a lodging for. A total of 10 popular countries are labeled using a shortform (for example `'ES'` for Spain), while an 11th category called `'other'` encompassed all non-popular countries. Finally, a 12th category labeled `'NDF'` (No Destination Found) indicated that the user did not end up booking lodging using the site.

To create synthetic data for this prediction problem, the SDV synthesized new users, and then synthesized new sessions for those users. It was not necessary to synthesize `Countries` because it was the parent table of the table containing the prediction problem. It was also unnecessary to synthesize `age_gender_bkts` because it was not a child of `Users`.

Rossmann

Kaggle's Rossman Store Sales dataset was another competition [11] based on history sales data for different stores in the franchise [10]. The Rossmann franchise is one of the largest drug store companies in Germany, and the dataset provided information about each individual store, as well as weekly details about it. This is described in Figure 6-4.

Store

Store	Primary Key
SoreType	Categorical
Assortment	Categorical
CompetitionDistance	Number
CompetitionOpenSinceMonth	Number
CompetitionOpenSinceYear	Number
Promo2	Categorical
Promo2SinceWeek	Number
Promo2SinceYear	Number
PromoInternal	Categorical

←

Train

Store	Foreign Key
DayofWeek	Categorical
Date	Datetime
Sales	Number
Customers	Number
Open	Categorical
Promo	Categorical
StateHoliday	Categorical
SchoolHoliday	Categorical

Figure 6-4: The schema from the Rossmann Store dataset. Each store is a different store location of the Rossmann franchise, and each row in `Train` corresponds to a particular day in the store.

The prediction problem was the `'Sales'` field in the `Train` table, that represented the total revenue made by the store in that day. Because this was a continuous variable, it was discretized into 8 bins.

Creating a synthetic version of the data meant synthesizing different stores first, and then synthesizing the rows in `Train` for each of those stores.

Telstra

The final dataset was from a Kaggle competition with a dataset from Telstra [17], [16]. Telstra is a telecommunications service from Australia that provides mobile phones and broadband internet. The layout of the dataset is described by Figure 6-5.

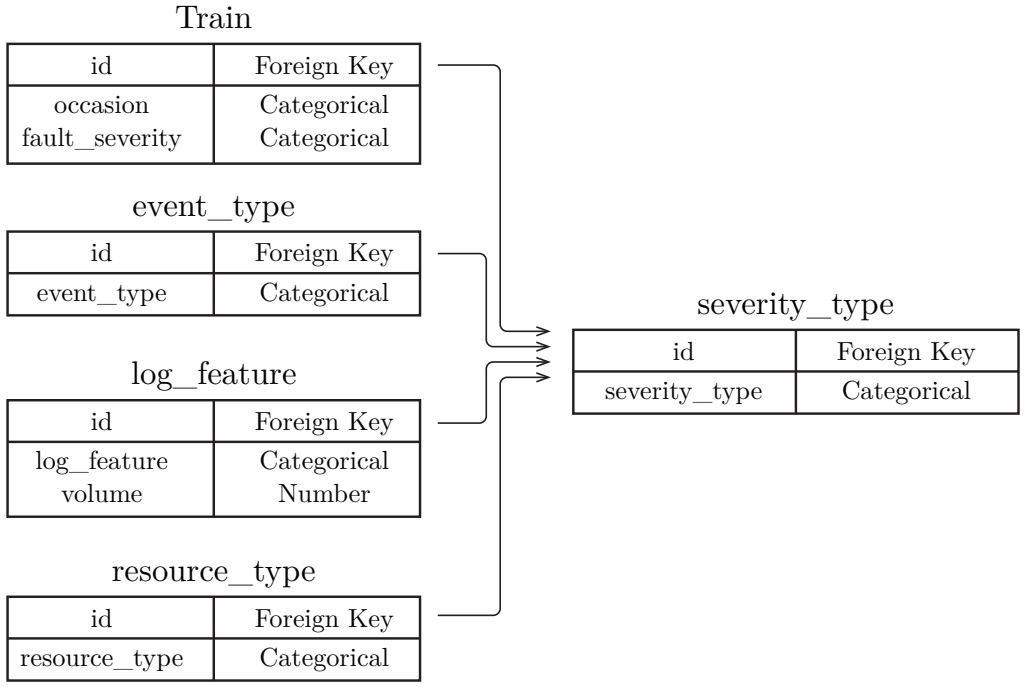

Figure 6-5: The schema for the Telstra dataset. Each column named of `'id'` represents a location and time. The information is split up by tables with meta-information about the event, log, resources, and severity of a possible network outage.

The prediction problem is to classify the `'fault_severity'` column of the `Train` table. This is either `'0'` for no network outage, `'1'` for a few outages, or `'2'` for many outages.

To create a synthesized version for this data, it was only necessary to synthesize new rows of the `Train` table, because this table had no children.

6.2.2 Crowdsourcing Experiment Setup

For each dataset, the SDV created four versions of data, each a condition for a within-subjects experiment with hired data scientists. These conditions were:

1. **Control**: The subject is presented with the original version of the dataset.
2. **No Noise (Synthesized)**: The subject is presented with the synthesized output from the SDV's algorithm.

3. **Table Noise (Synthesized)**: The subject is presented with synthesized noised output from the SDV's algorithm. The noise is introduced by taking every covariance value, σ_{ij}, $i \neq j$ and halving it, effectively reducing the strength of the covariance.

4. **Key Noise (Synthesized)**: The subject is presented with synthesized noised output from the SDV's algorithm. The noise is introduced by randomly sampling a primary key for the foreign key relation instead of performing an inference.

Subjects with some experience analyzing data were hired for the experiment. These data scientists were assigned into one of four groups, which determined the versions of the datasets. This is specified by Table 6.3.

Group	Biodegradability	Mutagenesis	Airbnb	Rossmann	Telstra
0	control	table noise	key noise	no noise	control
1	no noise	key noise	control	table noise	no noise
2	table noise	control	no noise	key noise	table noise
3	key noise	no noise	table noise	control	key noise

Table 6.3: The versions of each dataset that were available to each experiment group. While this setup may be biased to some ordering effects, it ensures that a single group receives differently synthesized versions of different datasets.

All subjects were given a briefing in which they were told to write complex features for each of the datasets. We used Feature Factory [18] as the interface for conducting the experiment. Each dataset was exposed to the subjects as a separate iPython notebook. The notebook contained some background information about the domain, as well as access to a variable `dataset` that contained a list of table objects belonging to the dataset.

Subjects were not told which version of the data they were given. In order to test their features, subjects were provided with a method called `cross_validate` that automatically computed their features and returned an accuracy score based on their

version of the dataset. Finally, when subjects finished and submitted their features, Feature Factory saved the script, which we used for our analysis.

6.2.3 Results

At the time of this writing, the experiment is still in progress. So far, we have collected data from 15 different subjects, each of whom has completed 3 or 4 out of the 5 datasets. As a result, we are still missing data to perform extensive analysis for the Rossmann and Telstra datasets, so we focus on the other three.

When interpreting the submitted features, the three questions we wished to explore were:

- Did the synthesized data provide adequate feedback about how well the features would predict real data?
- Was there a difference in the crowd's performance in terms of predictive accuracy when they were given original vs. synthetic data?
- On a subjective note, did the data scientists using synthetic data feel confused?

Answering each question required different analysis from the submitted features.

Adequate Feedback

If data scientists were to use synthesized data for all their work, they must be reasonably certain that what they produce will perform just as well on the real version of the data. This means that the synthesized data should provide accurate feedback about the data scientist's work.

Typically, data science or feature engineering is an iterative process. Data scientists write features, check their performance, and attempt to improve based on the feedback. While we don't expect a direct mapping between performance of a feature on synthesized data to real data, we do expect the following: Let f_1 and f_2 be two different sets of features. Let A_s be the accuracy function on the synthetic dataset, and A_r be the accuracy function on the real data. If $A_s(f_1) \leq A_s(f_2)$, then it should

be the case that $A_r(f_1) \leq A_r(f_2)$. This means that if the data scientist submits f_2 instead of f_1, it will improve accuracy for synthetic and real data.

To test for this, we performed the following steps for each subject's submitted work:

1. Let c be the original control dataset. Let v be the version of the dataset that this subject was given.
2. If $c \neq v$, split v into a train set and validation set.
3. Use the train split to create a model using the submitted features, f.
4. Record the accuracy of f on the validation split. This is the synthetic score, $A_s(f)$.
5. Now use f to predict values in the original datset, c. Record the accuracy as the real score, $A_r(f)$.

Thus, for every subject who was not in the control group, we calculate a synthetic score, $A_s(f)$, and a corresponding real score, $A_r(f)$ for their features. The synthetic score simulates the data scientist's estimate of how accurate their work is. The real score is the actual accuracy.

Hypothesis: There is a strong correlation between the synthetic score and the real score for each subject's work. A generally positive correlation means that the synthesized datasets give feedback that reasonably estimates the correct feedback. This implies that the synthesized data can be used successfully for data science.

Figure 6-6 shows a scatter plot of the synthetic score vs. the real score for all subjects across all datasets they submitted answers for. A linear regression test on the data shows that the correlation is statistically significant ($r^2 = 0.687$, $p < 0.001$). Furthermore, the slope is 0.970 and y-intercept is 0.034, indicating that the synthetic score closely predicts the test score.

Afterwards, we performed a 2-sample paired t-test on each submission's synthetic and accuracy score. The result showed that there was no significant difference between the two scores ($t = 0.812$, $p = 0.427$). This enables to conclude that $A_r(f) \approx A_s(f)$, a tighter constraint than we had initially set out to prove. It supports our belief that

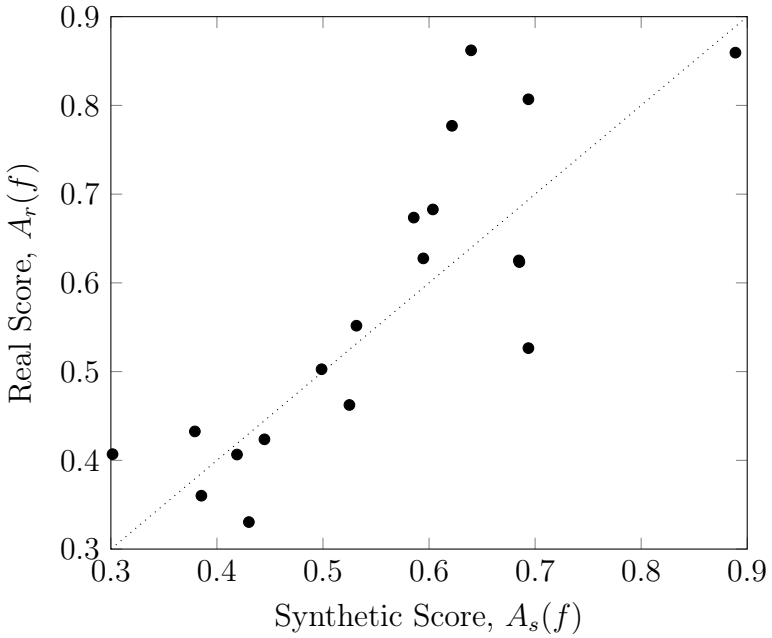

Figure 6-6: Results for synthetic score on the synthesized datasets vs. real score on the original dataset. The dotted line provides a reference for where synthetic score is exactly equal to the real score. The control group's data is not included in this plot.

synthetic data provides adequate feedback to the data scientist. We can conclude that the data scientist can use the synthetic data to reasonably gage the usefulness of their work.

Accuracy

Another concern is how a synthesized version of the dataset might affect the overall accuracy of the model, as compared to the original dataset. Even if the generated data provided consistent feedback to the subject, it would be of no use if the data scientist wasn't able to achieve high accuracy.

To test this, we performed a train-validate split on all versions of the dataset to train models for each subject. To calculate the overall test score, we used an external test set that was unavailable to the subjects.

- Non-Kaggle Data: We had taken out 25% of the data prior to inputting the data in the SDV. This 25% becomes a test set because no data scientist in the control group had access to it, and no synthetic data was created from it.

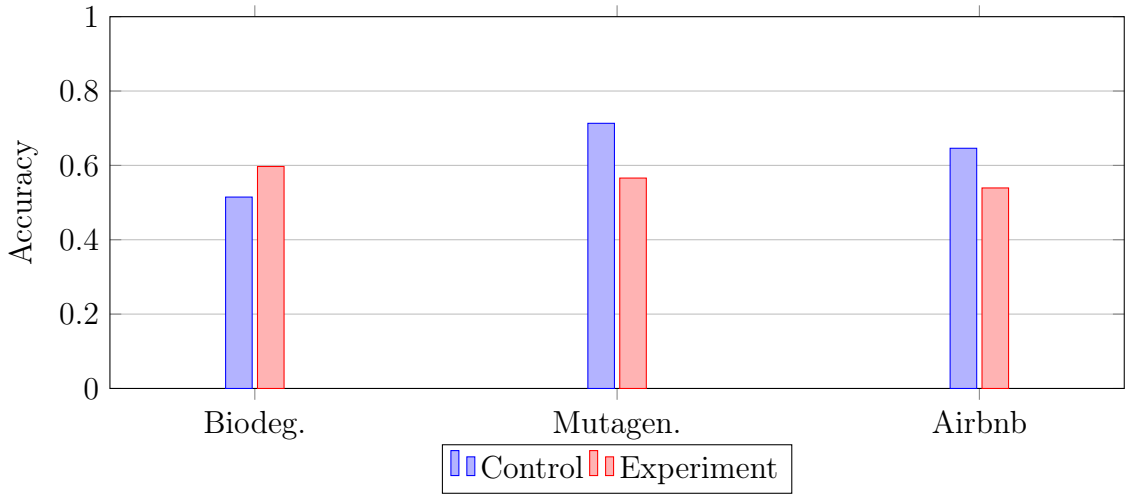

Figure 6-7: A graph containing the median accuracies of the control vs. experimental groups for all datasets.

Dataset	t-statistic	p-value
Biodegradability	-0.558	0.701
Mutagnesis	1.15	0.139
Airbnb	0.372	0.364

Table 6.4: An accuracy comparison for control vs. non-control groups, broken down by each dataset. Results from the t-test, as well as a one-sided p-value are provided for each dataset.

- Kaggle Data: Kaggle provides an online test set for their competitions. Kaggle does not provide the solutions, so we submitted the subject's predictions and recoded the score that Kaggle reported.

We compared the test scores between the control groups, and the other groups.

Hypothesis: The test scores between the control group and the other groups will not be different. This would show that data scientists can use synthetic data to submit work that is just as accurate as work done using the real data.

Figure 6-7 illustrates the differences in accuracies per dataset. Table 6.4 shows preliminary results from a 2-sample independent t-test performed between the control and experiment groups.

Overall, we find that there is no statistically significant difference between the accuracy scores of subjects with control data and subjects with generated data. This

confirms our belief that scientists can easily work with synthesized data as they can with control data. It remains to be seen how the levels of noise in the synthesized data affect the accuracy. By running the experiment for longer and collecting more features, we will be able to perform this analysis at a future time.

Subject Findings

Finally we consider subjective feedback provided to us by the data scientists. In particular, we observed the questions different subjects asked to see if they were confused by the data they were provided.

A large majority of the questions were about technical issues with Feature Factory or the experimental setup that were unrelated to the synthesized data. Apart from asking how to register and submit their features, many subjects were also unhappy that they did not have access to the target column's data. We purposefully designed the experiment in this way so that users could focus on writing features without knowing what the actual answers are. However, in the future, perhaps we would consider allowing data scientists to see the target column of the training data, in order to mimic the workflow more closely.

Some users were confused about the relations that existed between the tables. One subject in particular did not understand how two columns of the `bond` table could be foreign keys to the `atom` table in the Mutagenesis dataset (Figure 6-2). Other users defaulted to writing features from the target table only. We explicitly encouraged the subjects to join the tables to explore the entire dataset.

Only 1 question was related to the actual values in the dataset. A subject in group 3 indicated that the `ages` column in the `Users` table of the Airbnb dataset (Figure 6-3) had unrealistic ages. This user's data was synthesized with table noise. However, upon closer inspection, it appears that the original data for Airbnb also had unrealistic data in the `ages` column (max age was 2004). The SDV synthesized data within the correct bounds when compared to the control data.

However, we did realize that the SDV's bounds will not always be correct if we are synthesizing children from a noised parent table. This is because the parent holds

the *min* and *max* values of their children's columns. If the parent is noised, then the *min* and *max* may represent unrealistic data.

Ultimately, we found that the SDV successfully modeled each of the relational datasets, and used the generative models to synthesize data that data scientists could realistically work with. Though the experiment is still in progress, our preliminary results show promise in using the SDV for data science purposes.

Chapter 7

Conclusion

7.1 Key Findings

The SDV was successful for each of our goals for generalizability, usability, and accuracy. Here, we provide key takeaways from each of these areas.

The SDV can be applied generally to a variety of relational datasets. During our experimentation phase, we applied the SDV to Biodegradability, Mutagenesis, Airbnb, Rossmann, Telstra, and industrial datasets. The SDV was able to model the relational data automatically for each of these datasets, with no changes to the code.

The SDV can synthesize data at a variety of granularities for different test purposes. Our work with the our sponsor, Mutability, and Rossmann datasets required the SDV to synthesize the entire database, with all the tables and their corresponding foreign key relations. The Biodegradability dataset required the SDV to synthesize all tables except for one (the `Group` table), while ensuring that all key relations between existing and synthesized tables were accurate. Finally, the Telstra and Airbnb datasets required the SDV to synthesize a single table whose foreign keys accurately referenced their parents. The versatility of SDV shows that it can be adapted to many types of problems.

The synthetic output from SDV can replace original data for the purposes of data science. As of May 18, 2016, our results indicate that data scientists

were able to work as effectively with the synthetic output as they were with the original data. In particular, a regression between the cross validation and test score showed that the synthetic data gave the correct feedback to data scientists when validating their models ($p < 0.001$). A comparison in overall accuracies between the original and synthetic data showed no statistically significant effects between the type of data and the data scientist's ultimate performance on the test set.

7.2 Contributions

In this thesis, we:

1. Designed *CPA*, an approach that builds a generative model for a table that has external table dependencies. When layered recursively, this forms the *RCPA*, which models an entire relational database.
2. Created a method for inference and synthetic data generation across multiple tables. This uses covariance update rules for generative models as its foundation.
3. Implemented the *SDV*, an end-to-end system that allows users to build generative models for relational databases, and use the model to synthesize data.
4. Demonstrated that the SDV meets its goals for usability and generalizability by using it to model 6 different datasets from a combination of sources: our sponsor, the relational database repository, and Kaggle.
5. Evaluated the SDV's ability to synthesize data for sample databases by working a real-world complex relational database from our sponsor. Demonstrated that the SDV synthesizes data that be used for testing.
6. Formulated metrics to quantify how much synthesized data affects the ability to solve a prediction problem.
7. Performed experiment using Feature Factory, and analyzed submitted features to demonstrate that synthetic output from SDV:
    - gives effective feedback regarding its application to real data
    - does not interfere with the data scientists' ability to make accurate predictions

- does not produce confusing data that impedes the data scientists' progress

We conclude that the SDV successfully builds generative models for relational databases, and is a viable solution for synthesizing data.

Bibliography

[1] Airbnb. https://www.airbnb.com/. Accessed: 2015-05-16.

[2] Kaggle airbnb new user bookings. https://www.kaggle.com/c/airbnb-recruiting-new-user-bookings. Accessed: 2015-05-16.

[3] Hendrik Blockeel, Sašo Džeroski, Boris Kompare, Stefan Kramer, and Bernhard Pfahringer. Experiments In Predicting Biodegradability. *Applied Artificial Intelligence*, 18(2):157–181, 2004.

[4] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991.

[5] Kaggle. https://www.kaggle.com/. Accessed: 2015-05-16.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[7] R.J.A. Little and D.B. Rubin. *Statistical Analysis with Missing Data*. Wiley, 2002.

[8] F. J. Massey. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951.

[9] Jan Motl and Oliver Schulte. The CTU prague relational learning repository. *CoRR*, abs/1511.03086, 2015.

[10] Rossmann mein drogeriemarkt. https://www.rossmann.de/verbraucherportal.html. Accessed: 2015-05-16.

[11] Kaggle rossman store sales. https://www.kaggle.com/c/rossmann-store-sales. Accessed: 2015-05-16.

[12] L. Rüschendorf. *Mathematical Risk Analysis*, chapter 1. Springer, 2013.

[13] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 01 2016.

[14] Latanya Sweeney. Uniqueness of Simple Demographics in the U.S. Population, 1000.

[15] Latanya Sweeney. K-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, October 2002.

[16] Telstra.com. https://www.telstra.com.au/. Accessed: 2015-05-16.

[17] Telstra network disruptions. https://www.kaggle.com/c/telstra-recruiting-network. Accessed: 2015-05-16.

[18] Kalyan Veeramachaneni, Kiarash Adl, and Una-May O'Reilly. Feature factory: Crowd sourced feature discovery. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*, L@S '15, pages 373–376, New York, NY, USA, 2015. ACM.