gretel™ | CASE STUDY

Creating Synthetic Time Series Data for Global Financial Institutions

JANUARY 17, 2022

CONTRIBUTORS

Amy Steier
Alex Watson
Lipika Ramaswamy
Daniel Nissani
Kendrick Boyd

Table of contents

Introduction

In this study, we discuss the creation of high-quality synthetic time-series datasets for one of the largest financial institutions in the world, and the methods we designed to assess the accuracy and privacy of our models and data. The temporal, ordered nature of time series data can help track and forecast future trends, which unsurprisingly, has enormous utility for business planning and investing. However, due to regulations and the inherent security risks that come with sharing data between individuals and organizations, much of the value that could be gleaned from it remains inaccessible. Here, Gretel's work demonstrates that synthetic data can help close this gap while preserving privacy. By generating synthetic time-series data that are generalizable and shareable amongst diverse teams, we can give financial institutions a competitive edge and the power to explore a whole new world of opportunities.

Developers can test our methods by opening up our example Colab Notebook, clicking "Run All", and entering your API key to run the entire experiment, or by following along with the 3-step process outlined in this document.

Follow along:
https://colab.research.google.com/github/gretelai/gretel-blueprints/blob/main/docs/notebooks/time_series_generation_poc.ipynb

GitHub:
https://colab.research.google.com/github/gretelai/gretel-blueprints/blob/main/docs/notebooks/time_series_generation_poc.ipynb

The Test Dataset

For this case study, the bank's data science team provided a time series dataset containing customer account balance information over time, which spans 6 columns and approximately 5500 rows. There is a time component in the `date` field, and a set of trending account balances that must be maintained across each `district_id`.

```
# Load timeseries example to a dataframe
import pandas as pd

def load_data(dataset: str) -> pd.DataFrame:
    df = pd.read_csv(dataset)
    return df.sort_values('date', inplace=True)

dataset = 'https://gretel-public-website.s3.amazonaws.com/datasets/credit-timeseries-dataset.csv'
train_df = load_data(dataset)
train_df
```

	net_amt	date	account_balance	credit_amt	district_id	debit_amt
1814	1100.0	1993-01-31	1100.0	1100.0	52	0.0
5392	0.0	1993-01-31	0.0	0.0	2	0.0
3888	200.0	1993-01-31	200.0	200.0	23	0.0
1509	0.0	1993-01-31	0.0	0.0	56	0.0
1106	5944.5	1993-01-31	5944.5	5944.5	62	0.0
...	...	...	...	...	...	...
5086	135240.1	1998-12-31	2044851.7	827582.8	3	692342.7
169	578835.9	1998-12-31	5873241.7	3070517.4	74	2491681.5
3228	63739.4	1998-12-31	1599160.1	525859.4	30	462120.0
4743	76175.8	1998-12-31	2266179.9	1086790.0	9	1010614.2
5039	143492.5	1998-12-31	1999796.3	855391.8	7	711899.3

5544 rows × 6 columns

Figure 1 - Sample of the training dataset

STEP 1

Create a pipeline to synthesize the time series dataset

First, we create a simple pipeline (see diagram below) that can be used to de-identify the data and then build a synthetic model to generate an artificial version of the same size and shape.

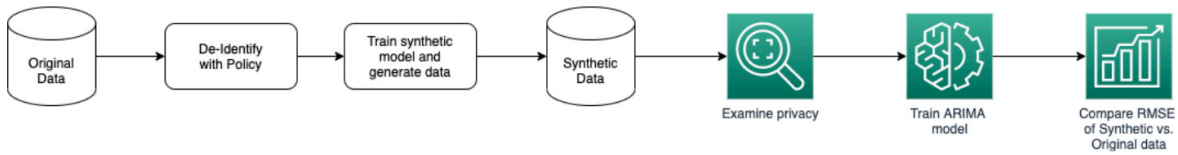


Figure 2 - Synthetic data pipeline

To de-identify the data, we use Gretel.ai's Transform API with the configuration policy below to randomly shift any dates and floating point values in the data. To ensure consistent shifts for each district, we define the `district_id` identifier as a seed value. This step provides strong initial guarantees that a synthetic model will not memorize repeated values in the data.

```
[ ]  # Gretel Transforms Configuration
     config = """
     schema_version: "1.0"
     models:
         - transforms:
             data_source: "__tmp__"
             policies:
                 - name: shiftnumbers
                   rules:
                     - name: shiftnumbers
                       conditions:
                         field_name:
                             - account_balance
                             - credit_amt
                             - debit_amt
                             - net_amt
                       transforms:
                         - type: numbershift
                           attrs:
                             min: 1
                             max: 100
                             field_name:
                                 - date
                                 - district_id
     """
```

We then train a synthetic model on the de-identified training dataset, then use that model to create a synthetic data set with the same size and shape of the original data. For this, we will use Gretel's "smart seeding" task, which conditions the model with the `date` and `district_id` attributes specified below and effectively asks the model to synthesize the remaining data.

```
#Set up the seed fields
seed_fields = ["date", "district_id"]

task = {
    'type': 'seed',
    'attrs': {
        'fields': seed_fields,
    }
}

# Fine tune model parameters. These are the parameters we found to work best.  This is "Run 20" in
the document
config['models'][0]['synthetics']['task'] = task

config['models'][0]['synthetics']['params']['vocab_size'] = 19
config['models'][0]['synthetics']['params']['learning_rate'] = 0.001
config['models'][0]['synthetics']['params']['epochs'] = 500
config['models'][0]['synthetics']['params']['dropout_rate'] = .5
config['models'][0]['synthetics']['params']['gen_temp'] = .8
config['models'][0]['synthetics']['generate']['num_records'] = train_df.shape[0]
```

STEP 2

Assess the accuracy of the data

Next, we need to test our model's accuracy, which starts with running a quick sanity check using a comparison of time series distributions for a district in our dataset. Here, Gretel's synthetic quality score report (viewable in the notebook) is helpful for assessing the model's ability to learn correlations in the data. Below, you can see our synthetics are in line with the original dataset. **Success!**

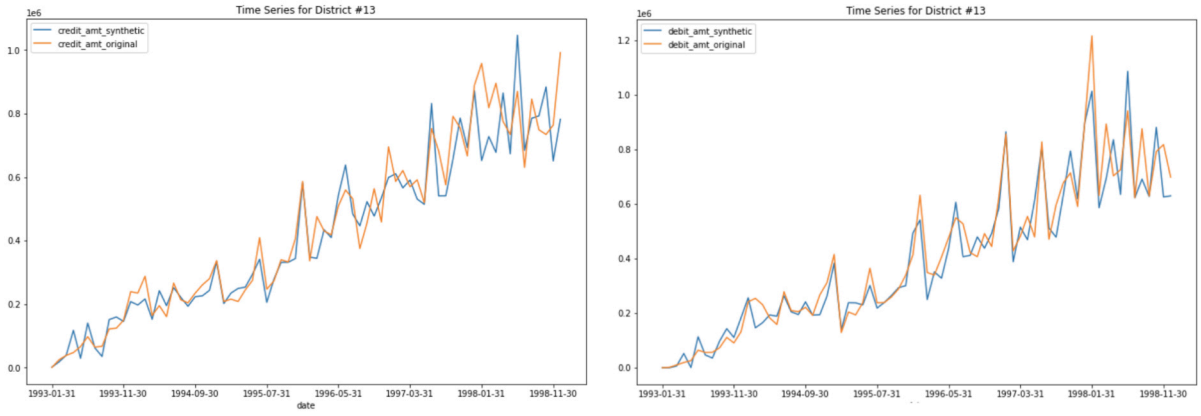


Figure 3 - Comparison of district distributions in the synthetic and original datasets

To tackle the tricky problem of assessing the quality of our synthetic time series dataset, the bank's team fitted an ARIMA model to both the synthetic and original datasets. This allowed us to measure how well each dataset captured seasonality in the data. (Note: our POC notebook uses the SARIMAX model in the statsmodels package, which includes a multiplicative seasonal component similar to the modified ARIMA model used by the bank's team.)

To train the SARIMAX models, we used district_id = 13 and ran an experiment on each available target variable (this generated four target variables). For evaluating our model's prediction accuracy i.e. the root-mean-square error (RMSE), we used the last year in the dataset (1998) as our validation. The available years prior were used for training the model. All other parameters in the SARIMAX model were left at their default settings, except for those provided by the bank's team. Namely, the order = (0,1,1) and the seasonal_order = (1,1,0,12). Lastly, we made sure to order the data by date for each run.

The images below are the RMSE results for each config used and the original dataset. The goal is to minimize RMSE. Here, it's notable that for many synthetic data runs, we outperformed the real data on at least one target variable. In particular, look at runs 17, 18, 19, and 20, where we controlled for the float precision of the columns before generating synthetic data. This control especially helped with runs 19 and 20, where we did significantly better than the real data on 3 out of 4 variables. Moreover, run 20 includes the suggestion of removing net_amt from the synthetic data generation, since it's a derivation of credit_amt and debit_amt, and thus adds needless complexity to the process. Runs 20Priv and 20SDKPriv both turned on privacy settings with all the improvements described above.

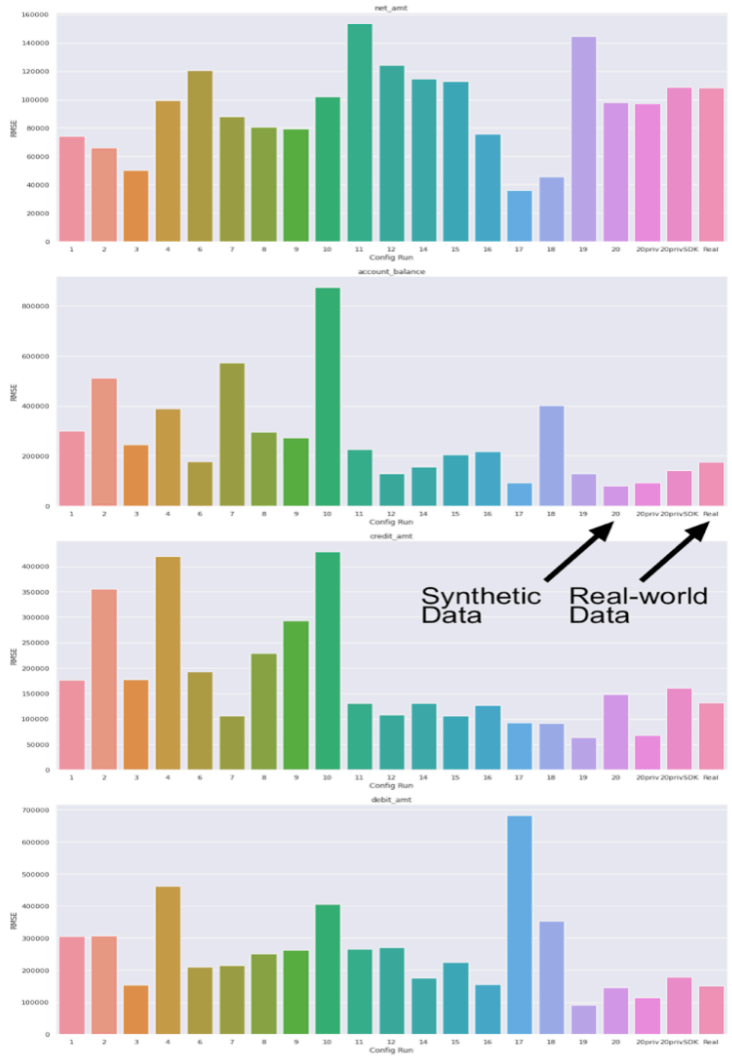


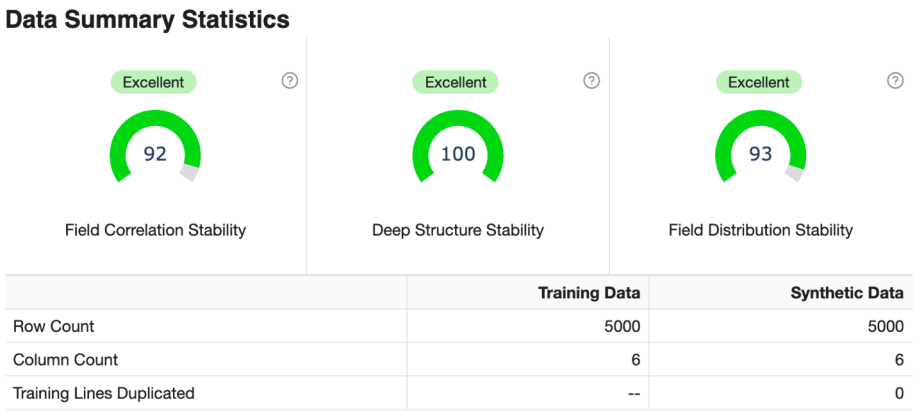
Figure 4 - RMSE results for each config run used and the original dataset.

The takeaway: optimal synthetic configurations can, in some cases, outperform the real-world data on our machine learning model by better minimizing RMSE!

STEP 3

Assess the privacy of the artificial data

Finally, after creating our synthetic model and dataset, we can assess the privacy of our artificial dataset. Let's compare our transformed and synthesized dataset to the original training dataset.



	Training Data	Synthetic Data
Row Count	5000	5000
Column Count	6	6
Training Lines Duplicated	--	0

Figure 5 - Statistical comparison of the synthetic and original training datasets

This notebook highlights a new feature in Gretel Synthetics called Privacy Filtering, which provides a form of armor to cover synthetic data weak points often exploited by adversarial attacks. For example, synthetics that are too similar to the original data can lead to membership inference attacks as well as attribute disclosure. Another serious privacy risk arises when you have synthetic 'outlier' records, particularly when they're similar to outlier training records. To combat both scenarios, we created Similarity and Outlier filters, which can both be dialed to a specific threshold based on the desired level of privacy. In this study, the use of Gretel's Similarity Privacy Filter removed all synthetic records that were duplicates of training records.

To take things a step further, one method for ensuring the privacy of every record in the original dataset is to enable `differential privacy,` where we add some statistical noise while training the synthetic data model. Intuitively, a model trained with differential privacy should not be affected by any single training example, or small set of training examples in its dataset. This process involves changing the optimizer that generates a well-fit synthetic model, but it requires a large number of records (typically >50k) to maintain the privacy of each record. For this reason, turning on differential privacy in the config wouldn't provide useful synthetic data while also maintaining a high standard of privacy.

Conclusion

In this POC, we successfully demonstrated that Gretel's synthetic data can be as accurate, and in some cases even surpass that of real-world data used for machine learning classification tasks, while also providing strong privacy guarantees required to allow sharing inside a financial institution.

About Gretel

Gretel.ai was founded on a privacy-first mission to equip developers with the ability to unlock innovation through safe, efficient collaboration with sensitive data. Gretel pioneered Privacy Engineering as a Service and designed a synthetic data tool suite based on their open sourced AI-based core. These tools make it easier and faster to generate privacy-preserving data that can be safely shared.

Gretel created easy to use, accessible APIs for developers and data practitioners to generate high quality synthetic data, classify and label, and transform and anonymize data – tools that quickly remove privacy-related bottlenecks, and accelerate business innovation for organizations in financial services, life sciences, healthcare, technology, gaming and other industries.