MyFaker - Fake Data Generator

https://github.com/HarikcUW/myfaker/ - Report by Hari Kishor Chintada

1. Introduction

For all machine learning (ML) projects data is required, and with the increase in privacy and security concerns, companies are restricting customer data usage. It is becoming harder and harder to get good quality data for development and testing. So, developers need to generate their own fake data to reflect real world scenarios.

The objective of this project is to build a package to generate random sample data defined by user

- Flexibility/Configurability: Users prefer flexibility and want to control how the data should be generated.
- Extensibility: Support custom data input: Each field needs different data, for example, medical data is different from sales data. So, it is helpful if the package supports bringing their own data.
- Manage relation between data columns
- Generate pandas' data frame: Simple to output or re structure in the required format

2. Use Cases

2.1. Targeted Users

This package is helpful for Students, programmers/developers, data scientists and analysts/researchers. They can use this library to generate data for their development and testing. Most data science work is of no use without data, similarly, need data for testing as well as to understand any existing package. Researchers often need different data sets to analyze various scenarios.

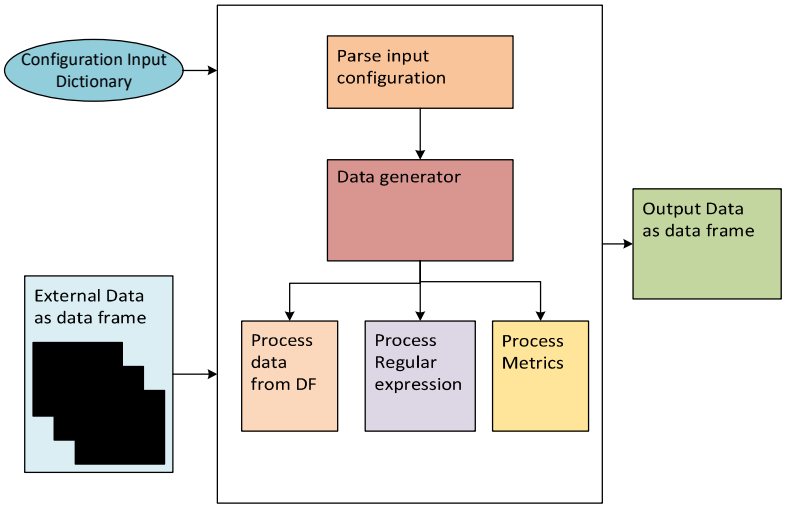
2.2. Use cases

- David, a student, would like to create a model using eCommerce sales data, but it is hard to find real credit card data for his experiment. He can use this package to generate his own data. He can specify the format of the card and other properties and generate thousands of records in no time.
- Mary, a data scientist, wants to analyze a model's performance and needs quality data. She can use this package to generate necessary data.

3. Software modules/components

3.1. Component diagram

The myfaker package consists of input, generator, output. Generator process input and generates output data frame using input configuration.



Input: There are 2 main components, configuration dictionary with json object to specify output schema and how the generator should generate each attribute/column. It supports bringing the user's own data as external data frames. Input configuration dictionary has the following items:

sourceType: What is the source for data, acceptable values: Dataframe, RegularExpression, Metrics

- values: dictionary list to get input, columns and its definition
    - for data frame – include actual data frame, column name, column rename (if different name required in output)
    - for regular expression – include column name, regular expression, prefix, suffix
    - for Metrics – column name, data type (either float or int), start and end values

Generator: It has 3 components, input configuration parser, data generator, data processors. Data processor provides random data from data frames or generates random values from regular expression or generates int/float random values.

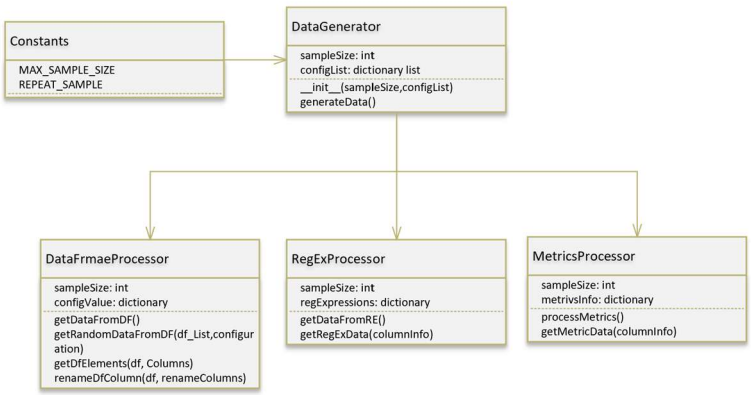
Output: Output is a data frame with requested data size and columns

3.2. Libraries used:

The following libraries/modules are used in this project.

- pandas: To use pandas data frame and operations on it
- random: To generate random value or sample
- rsts: To generate random string using simple regular expression

3.3. Class diagram



Input Configuration: This is the hardest component to design as end users need to prepare this configuration. Initially considered multiple input dictionaries and arguments using **args and **kwargs parameters. But as complexity grows it becomes hard to maintain these multiple parameters. So I decided to use a dictionary list with json elements. The myfaker package is designed to generate data using data from existing data frames, regular expression and int or float metrics. Each of these types requires different parameters, for example from data frame, select specific columns and need to think about how to avoid duplicate columns from different

data frames. Similarly for regular expression we need regular expression and for metrics supported range.

3.4. Software components

Constants.py: We have two constants, MAX_SAMPLE_SIZE to avoid overflow and avoid unexpected inputs (like -ve values) and REPEAT_SAMPLE to allow repeated sample values.

dataGenerator.py: A module to parse input configuration and generate data using appropriate process.

- DataGenerator class: This class has two variables sample size and user configuration
- generateData(self): Interface function that is used to prepare the data for each source type and combines the data to a single output data frame.

dataframeProcessor.py: Helps to process data from data frame.

- *DataframeProcessor* class: Users can input multiple data frames with data and select specific columns from each data frame, for example name and price from product data frame and name and iso code from country data frame. One challenge is handling the same attribute name from multiple data frames, so we must support renaming output attributes.
- *getDfElements(self, df, columnList)*: To generate random sample rows from data frame.
- *renameDfColumn(self, df, columnRenameDict)*:To rename list of columns in a data frame.
- *getRandomDataFromDF(self, df, columns)*:To process input json and prepare column list and rename column dictionary. This function will use getDfElements and renameDfColumn.
- *getDataFromDF(self)*: To loop through multiple data frames from the input.

regExProcessor.py: Process data using regular expression using rstr python package. It has additional features like generating values with some predefined prefix or suffix.

- *getRegExData(self, columnInfo)*: To generate random value using regular expression.
- *getDataFromDF(self)*: To loop through multiple regular expression columns from the input.

metricsProcessor.py: Used for generating metrics, like int or float values between given range.

- *getMetricData(self, metricInfo)*: To generate random int/flat value between given range.
- *processMetrics(self)*: To loop through multiple metric columns from the input.

4. Design decisions

The following design concepts were considered in this project implementation:

- **Object oriented design:** Created Data Generator class and defined necessary methods. Object oriented programming helped to organize code into multiple classes based on their functionality.
- **General purpose deep modules:** Defined helper class as general-purpose deep module. Depending on user input, the data generator performs various tasks and returns requested data.
- **Separation of concern:** The myfaker supports data generation using 3 different sources, input data frames, regular expression, and metrics. Since the data generation process is different for each type, individual classes for processing. Also created modules or functions to perform separate/independent tasks, for example separated input configuration process and actual data generation to support various configuration options.
- **Flexibility:** One of the key design considerations is to make it easy to use. Users need to provide a list of configurations as he/she controls how the data should be generated and output schema. After considering dynamic parameters, the list found that it will be hard to manage too many dynamic parameters. So I finally decided to use a dictionary list with json object elements. This will give flexibility to add new properties as well as supports backward compatibility.

5. Comparison with an existing library

I chose the publicly available faker library https://github.com/joke2k/faker to compare with myfaker library. Both are intended to generate fake data for development and validation. The main difference is flexibility, faker uses predefined data sets and myfaker uses users input data frames and supports regular expressions. There is some learning curve to using the faker library compared to my project, which completely generates data based on input whereas users must learn the structure of the faker data set. Another functional difference is faker library returns each attribute randomly, so it will not preserve relationships.

The myfaker library exposes a single function (interface) and the user can provide configuration (or metadata about required output) whereas the faker library exposes multiple classes related to data and the user must write code to get data. On the other hand, the faker library supports many languages and has rich sample data.

6. Extensibility

The myfaker library is designed such that it can be easily extended to support data generation from any other source. Developers can follow a similar workflow as defined in regular expression processor or data frame processor. They can define additional interfaces to simplify how users can pass configuration, something like user pass (categorical, categorical, categorical, int, int, float). Also, we can extend the output format easily as the library outputs pandas data frames and there are many modules available to transfer

pandas' data frames to other formats. Here I am listing a few ideas about how we can extend myfaker library.

- Extend to generate how many distinct values should generate for each categorical feature
- For Metrics – right now we generate random numbers, extend it to support any specific distribution
- Extend to use combination of data from data set and regular expression to derive data
- Extend to output data in different formats (Json, parquet, or csv format)

7. Usage example

Example#1: Generate random data using regular expression. In this scenario no input data is required.

```
from MyFaker.code.helper import DataGenerator

configList = [{'sourceType':'RegularExpression', 'values' : [ {'name' :'Featurs'
            ,'columns': [{'colName' : 'FirstName', 'prefix' : 'FirstName',  'sufix' : '', 'regExpression' : '[0-9]{2}'  }
                    ,{'colName' : 'UserEmail', 'regExpression' : '[A-Za-z]{6,10}[0-9]{2}@[a-z]{5}\d\.com' } ]}]}
           ,{'sourceType':'Metrics', 'values' : [ {'name' :'Featurs'
            ,'columns': [{'colName' : 'SalesQuantity', 'dataType' : 'int', 'startValue' : '10', 'endValue' : '40'  }
                    ,{'colName' : 'SalesAmount', 'dataType' : 'float', 'startValue' : '10', 'endValue' : '40'  } ]}]}
         ]

dg = DataGenerator(10,configList)
dfr = dg.processInput()
print(dfr)
```

Output:

```
    FirstName          UserEmail  SalesQuantity  SalesAmount
0  FirstName77    EdazLv72@yhopz7.com         40    15.711852
1  FirstName61   AuueXfjS62@foxfx2.com         12    22.351783
2  FirstName22   EuzYtEDx98@golia4.com         31    37.666205
3  FirstName47   tiJapfiSG18@dzkix6.com         25    11.626575
4  FirstName44  nzHOYMmrnt64@wxzzm0.com         10    14.879003
5  FirstName43    BacxPA97@hgftb6.com         17    27.401190
6  FirstName38    iUcXaW58@duzvu8.com         33    28.602415
7  FirstName95   RMGamxKL56@cguxl0.com         31    33.526898
8  FirstName63   FdzygrzbF49@fpueb5.com         26    28.541427
9  FirstName94    YHDtLYXs56@ibgbb3.com         19    18.988063
```

8. Summary

Data is required for almost all projects and often developers need to generate data themselves. The myfaker package can help to generate fake data using either from another data frame and regular expression.

9. References

- https://github.com/leapfrogonline/rstr
- https://github.com/joke2k/faker