PharmaSUG 2025 - Paper XX-###

Generating Synthetic Clinical Trial Data with AI: Methods, Challenges, and Insights

Max Ma, Weijie Yang, Everest Clinical Research Corporation;
Emily Ma, University of Toronto

ABSTRACT

In the clinical trial industry, the ability to generate realistic synthetic data is essential for programming preparation, validation, data simulation, and training. Artificial Intelligence (AI) offers powerful tools to achieve this by generating structured and meaningful synthetic data that aligns with predefined specifications. This paper discusses the use of Python and OpenAI's API to generate synthetic clinical trial data, utilizing the Pydantic library to enforce structured outputs. A comprehensive workflow is presented, covering the entire process from specification preparation to exporting data for Electronic Data Capture (EDC) systems or further analysis.

INTRODUCTION

To expedite clinical trial submissions, sponsors often aim to prepare programming for SDTM, ADaM datasets, and TFLs before the actual study data becomes available. However, creating dummy data for these purposes can be both time-consuming and tedious. Can Artificial Intelligence (AI) help streamline this process? As is often the case with current transformer-based large language models (LLMs), structured outputs are a common challenge. In this paper, we demonstrate how the OpenAI GPT model, in conjunction with the Pydantic package, can be leveraged to efficiently generate structured clinical trial data. We discuss the challenges of this approach and offer insights into how they can be addressed.

METHOD FOR GENERATING STRUCTURED DATA

Generating structured data involves several steps, which are illustrated in the following workflow:

Figure 1 Data Generation Workflow

DEVELOP SYNTHETIC DATA GENERATION SPECIFICATIONS (SDGS)

In the clinical trial data management and analysis process, Electronic Data Capture (EDC) systems are often used to collect clinical study data. Once the study database is set up in the EDC system, attributes of the Case Report Form (CRF) data can be exported. For instance, in iMedidata RAVE, the Architect Loader Spreadsheet (ALS) file can be exported, which includes all CRF form and field attributes for the study. The "Fields" tab in the ALS serves as an ideal starting point for defining the data generation specification.

To enable the LLM model to generate values as expected, an additional "Prompt" column can be added to the specification. This column contains descriptions of the fields, their constraints (e.g., value ranges, code lists), and dependencies on other fields. These prompts will be further illustrated in examples later in this paper. Below is an example of the data generation specification:

Form	Field	Label	Prompt
DM	BRTHDAT	Birth Date	Date of Birth, in format captial dd MMM yyyy
DM	AGE	Age	Age between 18 and 80, a numeric value in the format 3, derived from BRTHDAT based on today's date
DM	AGEU	Age Unit	Age Units, Text coded with AGEU, the value must be in ('DAYS', 'HOURS', 'MONTHS', 'WEEKS', 'YEARS'), 'DAYS' has a 0.0% chance, 'HOURS' has a 0.0% chance, 'MONTHS' has a 0.0% chance, 'WEEKS' has a 0.0% chance, 'YEARS' has a 100.0% chance
DM	SEX	Sex	Sex, Text coded with SEX1, the value must be in ('F', 'M'), 'F' has a 50.0% chance, 'M' has a 50.0% chance
DM	CHILDPOT	Is the subject of childbearing potential?	Childbearing Potential, the value must be in ('Y', 'N') for Female, 'NA' for male
DM	CHILDPOTRSN	Reason not childbearing	Reason not childbearing, must be generated for CHILDPOT=N only, the value must be in ('Post-menopausal', 'Surgically sterile', 'Other') for female,'Post-menopausal' should be generated for female between the ages of 45 and 55, blank for male
DM	CHILDPOTSP	If Other, Specify	If Other Specify, generate it only the corresponsing item is 'Other', free text up to 200 characters long
DM	ETHNIC	Ethnicity	Ethnicity, Text coded with ETHNIC, the value must be in ('HISPANIC OR LATINO', 'NOT HISPANIC OR LATINO', 'NOT REPORTED', 'UNKNOWN'), 'HISPANIC OR LATINO' has a 2.0% chance, 'NOT HISPANIC OR LATINO' has a 97.0% chance, 'NOT REPORTED' has a 0.5% chance, 'UNKNOWN' has a 0.5% chance

GENERATE PYTHON CODE USING SAS

Once the SDGS is prepared, we can use SAS to read the specification and automatically generate the corresponding Python programs. The process of converting the specification to Python is straightforward, so we will not go into detail here. Instead, we will focus on the generated Python code using three forms: Demographics (DM), Adverse Events (AE), and Vital Signs (VS) as examples. Below is the project structure for the demonstration:

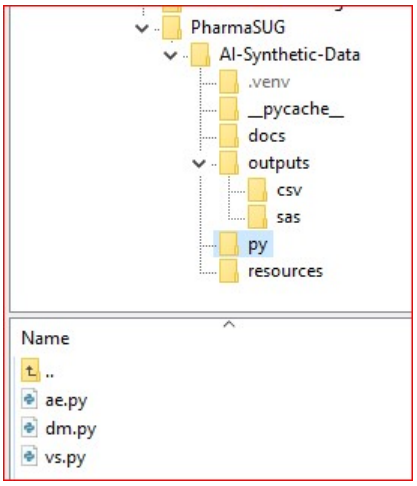


Figure 2 Project Structure

Here is the main function of the dm.py:

```
# Extract structured data from natural language
def gen_data(Topic: str, Nrecords: str) -> OutList:
    prompt = f"Generate exactly {Nrecords} {Topic} records."
    try:
      # Make your OpenAI API request here
      response = client.chat.completions.create(
        model="gpt-4o-mini",
        max_retries=3, # Retry the request 3 times
        response_model=OutList,
        messages=[
           {"role": "system", "content": "You are a clinical expert specializing in the
           generation of high-quality synthetic data for clinical trial. Your task is
           to create synthetic data that strictly conforms to the provided
```

```
            specifications for each field, ensuring realism and logical consistency. The
            generated data must be accurate, plausible, and appropriate for rigorous
            testing and validation purposes."},
           {"role": "user", "content": prompt}
        ],
        temperature=0.7,
        #top_p=0.1,
        #seed=8
      )
    except OpenAIError as e:
      # Handle all OpenAI API errors
      print(f'Error: {e}')
    records=vars(response).get('outlist')
    df=pd.DataFrame([vars(s) for s in records])
    print(df)
    return df
```

Program 1. Function to Call OpenAI API with Parameters

In the code above, the function calls the OpenAI API with predefined system messages and constructs the user prompt based on the input parameters. The model "gpt-4o-mini" is used. The other hyperparameters are explained below:

- max_retries: Specifies the maximum number of times the API will attempt to retry a failed request automatically.
- temperature: Controls the randomness of the output by adjusting the probability distribution of possible completions. Set a low temperature for tasks requiring precise answers (e.g., coding) and a high temperature for creative tasks (e.g., storytelling, data simulation, etc.).
- top-p: An alternative to temperature that limits the response to only the most probable tokens whose cumulative probability is at or below top_p. The top_p parameter is commented out in this demo.
- seed: Sets a fixed starting point for the random number generator used in generating outputs, ensuring reproducibility of results. This is newly added parameter in the openAI API, still in beta testing, unstable, and not used in this demo.

Another important parameter is response_model, which is not a standard API parameter. It is part of the Python library Instructor, designed for working with structured outputs from large language models (LLMs). The OutList object is defined using the BaseModel and Field classes from the Pydantic library, as shown below:

```
class Metadata(BaseModel):
    BRTHDAT: str = Field(..., description="Date of Birth, in format capital dd MMM yyyy")
    AGE: int = Field(...,  description="Age between 18 and 80, a numeric value in the
         format 3,  derived from BRTHDAT based on today's date.")
    AGEU: str = Field(..., description="Age Units, Text coded with AGEU, the value must
          be in ('DAYS', 'HOURS', 'MONTHS', 'WEEKS', 'YEARS'), 'DAYS' has a 0.0% chance,
          'HOURS' has a 0.0% chance, 'MONTHS' has a 0.0% chance, 'WEEKS' has a 0.0%
          chance, 'YEARS' has a 100.0% chance")
    SEX: str = Field(..., description="Sex, Text coded with SEX1, the value must be in
         ('F', 'M'), 'F' has a 50.0% chance, 'M' has a 50.0% chance")
    CHILDPOT: str = Field(..., description="Childbearing Potential, the value must be in
              ('Y', 'N') for Female, 'NA' for male.")
    CHILDPOTRSN: str = Field(..., description="Reason not childbearing, must be generated
                 for CHILDPOT=N only, the value must be in ('Post-menopausal',
                'Surgically sterile', 'Other') for female,'Post-menopausal' should be
                 generated for female between the ages of 45 and 55, blank for male")
    CHILDPOTSP: str = Field(..., description="If Other Specify, generate it only the
                corresponding item is 'Other', free text up to 200 characters long.")
    ETHNIC: str = Field(..., description="Ethnicity, Text coded with ETHNIC, the value
            must be in ('HISPANIC OR LATINO', 'NOT HISPANIC OR LATINO', 'NOT REPORTED',
            'UNKNOWN'), 'HISPANIC OR LATINO' has a 2.0% chance, 'NOT HISPANIC OR LATINO'
            has a 97.0% chance, 'NOT REPORTED' has a 0.5% chance, 'UNKNOWN' has a 0.5%
            chance")
```

```
    RACE_WHITE: str = Field(..., pattern=r"^(0|1)$", description="Race (White), a numeric
                value in the format 1, the value must be in ('1 = Checked', '0 = Not
                Checked'), '1' has a 40.0% chance")
    RACE_ASIAN: str = Field(..., pattern=r"^(0|1)$", description="Race (Asian), a numeric
                value in the format 1, the value must be in ('1 = Checked', '0 = Not
                 Checked'), '1' has a 10.0% chance")
    RACE_BLACK: str = Field(..., pattern=r"^(0|1)$", description="Race (Black or African
                American), a numeric value in the format 1, the value must be in ('1 =
                Checked', '0 = Not Checked'), '1' has a 10.0% chance")
    RACE_AMERIND: str = Field(..., pattern=r"^(0|1)$", description="Race (American Indian
                   or Alaska Native), a numeric value in the format 1, the value must be
                   in ('1 = Checked', '0 = Not Checked'), '1' has a 10.0% chance")
    RACE_NHAWII: str = Field(..., pattern=r"^(0|1)$", description="Race (Native
                    Hawaiian Pacific Islanders), a numeric value in the format 1, the
                    value must be in ('1 = Checked', '0 = Not Checked'), '1' has a 10.0%
                    chance")
    RACE_NR: str = Field(..., pattern=r"^(0|1)$", description="Race (Not Reported), a
                    numeric value in the format 1, the value must be in ('1 = Checked', '0
                    = Not Checked'), '1' has a 10.0% chance")
    RACE_OTHER: str = Field(..., pattern=r"^(0|1)$", description="Race (Other), a numeric
                value in the format 1, the value must be in ('1 = Checked', '0 = Not
                Checked'), '1' has a 10.0% chance")
    RACEOTH: str = Field(..., description="If Other Specify, must generate it if the
             field RACE_OTHER is '1', free text up to 200 characters long, for example:
             'Caribbean', 'Aboriginal Australian', 'Samoan' or 'Tongan', etc., if it's
             null and RACE_OTHER='1', set it to 'Unknown")

class OutList(BaseModel):
    outlist: List[Metadata] = Field(..., description="Multiple records")
```

Program 2. Define Structured Outputs

In the code above, the field definition attributes are derived from the SDGS. The prompt in the SDGS is mapped to the description in the field definition. Below are explanations of some types of descriptions:

- Multiple choices: This is the most common field type in CRF forms. You need to explicitly specify all the choice options with predefined probabilities, if needed. For example, for **SEX**: If the protocol does not specify the gender distribution requirement, you can simply list the choices as 'F' or 'M' without probabilities. If the protocol requires a specific distribution, you can specify it in the description. For instance, "*the value must be in ('F', 'M'), 'F' has a 50.0% chance, 'M' has a 50.0% chance*".
- Input with constraints: If there are constraints for an input field, you must explicitly specify them. For example, for **AGE**, if the protocol defines the acceptable age range as 18–80 years, include this information in the description. For instance: *"Age between 18 and 80, a numeric value in the format 3, derived from BRTHDAT based on today's date".*
- Input with dependencies: If input values depend on values in another field, specify this clearly in the description with detailed instructions. For example, for the **CHILDPOTRSN** field, the description could be: *"Reason not childbearing, must be generated for CHILDPOT=N only, the value must be in ('Post-menopausal', 'Surgically sterile', 'Other') for female,'Post-menopausal' should be generated for female between the ages of 45 and 55, blank for male".*
- Edit Check for Data Integrity: To ensure data integrity, include edit checks in a field. For example, if the value selected is "Other," the corresponding "Other Specify" field must be populated. For instance, for **RACEOTH**, the description could be: *"If Other Specify, must generate it if the field RACE_OTHER is '1', free text up to 200 characters long, for example: 'Caribbean', 'Aboriginal Australian', 'Samoan' or 'Tongan', etc., if it's null and RACE_OTHER='1', set it to 'Unknown'".*

EXECUTE PYTHON CODE TO GENERATE SYNTHETIC DATA

With a registered OpenAI API key and all required Python dependency packages installed, the Python code can be executed to generate the data. The data can then be exported as a CSV file with a '|' delimiter for upload to the EDC system for testing or training purposes. Alternatively, the data can be exported as a SAS transport file (XPT) for further processing in SAS environment.

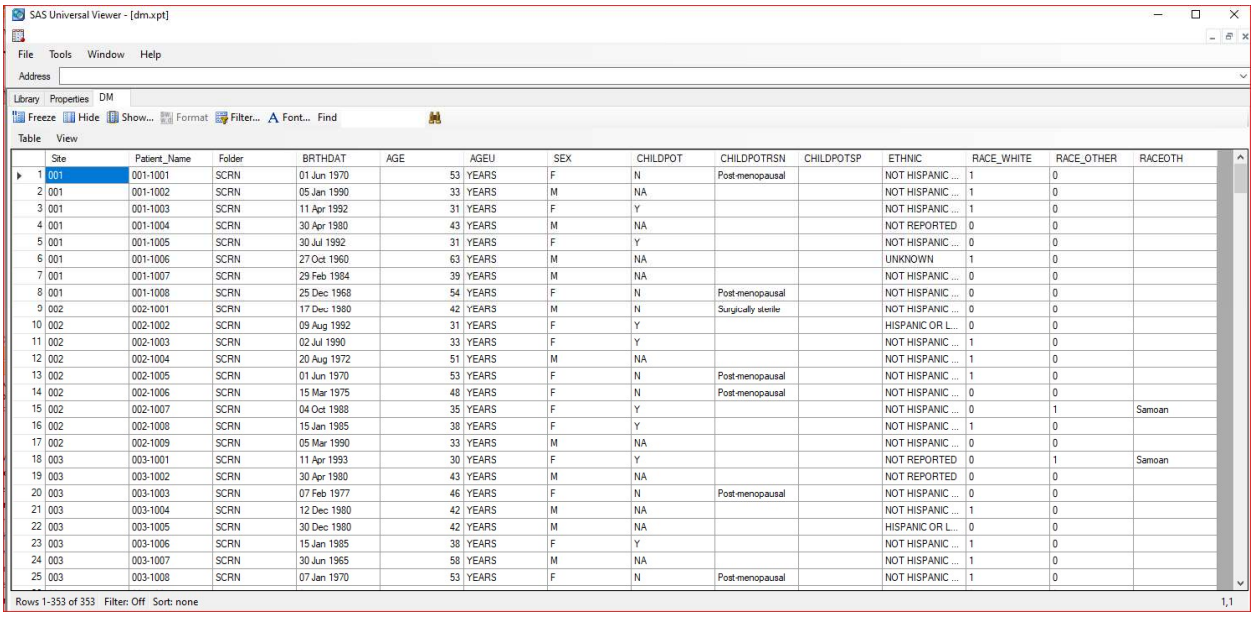
Below is a screenshot showing the execution of Python code in the terminal:

```
(.venv) [mma@python AI-Synthetic-Data]$ python ./py/dm.py
Generating: DM ...
Generating: 100 records for Demographics ...
Generated: 10 records for Demographics
Generated: 20 records for Demographics
Generated: 30 records for Demographics
Generated: 40 records for Demographics
Generated: 50 records for Demographics
Generated: 60 records for Demographics
Generated: 70 records for Demographics
Generated: 80 records for Demographics
Generated: 90 records for Demographics
Generated: 100 records for Demographics
    Site Patient_Name Folder  Record_position  Page_repeatnumber      BRTHDAT  ...  RACE_BLACK RACE_AMERIND RACE_NHAWII RACE_NR RACE_OTHER RACEOTH
0    001     001-1001   SCRN                0                  0  01 Jun 1970  ...           0            0           0       0          0
1    001     001-1002   SCRN                0                  0  05 Jan 1990  ...           0            0           0       0          0
2    001     001-1003   SCRN                0                  0  11 Apr 1992  ...           0            0           0       0          0
3    001     001-1004   SCRN                0                  0  30 Apr 1980  ...           0            0           1       0          0
4    001     001-1005   SCRN                0                  0  30 Jul 1992  ...           0            1           0       0          0
..   ...          ...    ...              ...                ...          ...  ...         ...          ...         ...     ...        ...     ...
348  046     046-1007   SCRN                0                  0  11 May 1992  ...           1            0           0       0          0
349  046     046-1008   SCRN                0                  0  29 Oct 1978  ...           0            0           0       0          0
350  046     046-1009   SCRN                0                  0  15 May 1960  ...           0            0           0       0          0
351  046     046-1010   SCRN                0                  0  25 Apr 1978  ...           0            0           0       0          0
352  046     046-1011   SCRN                0                  0  15 Jan 1985  ...           0            0           0       0          0

[353 rows x 21 columns]
(.venv) [mma@python AI-Synthetic-Data]$
```

Display 1. Execution of Python Code in Terminal

Below is a portion of the generated DM dataset in SAS XPT format:



	Site	Patient_Name	Folder	BRTHDAT	AGE	AGEU	SEX	CHILDPOT	CHILDPOTRSN	CHILDPOTSP	ETHNIC	RACE_WHITE	RACE_OTHER	RACEOTH
1	001	001-1001	SCRN	01 Jun 1970	53	YEARS	F	N	Post-menopausal		NOT HISPANIC ...	1	0	
2	001	001-1002	SCRN	05 Jan 1990	33	YEARS	M	NA			NOT HISPANIC ...	1	0	
3	001	001-1003	SCRN	11 Apr 1992	31	YEARS	F	Y			NOT HISPANIC ...	1	0	
4	001	001-1004	SCRN	30 Apr 1980	43	YEARS	M	NA			NOT REPORTED	0	0	
5	001	001-1005	SCRN	30 Jul 1992	31	YEARS	F	Y			NOT HISPANIC ...	0	0	
6	001	001-1006	SCRN	27 Oct 1960	63	YEARS	M	NA			UNKNOWN	1	0	
7	001	001-1007	SCRN	29 Feb 1984	39	YEARS	M	NA			NOT HISPANIC ...	0	0	
8	001	001-1008	SCRN	25 Dec 1968	54	YEARS	F	N	Post-menopausal		NOT HISPANIC ...	0	0	
9	002	002-1001	SCRN	17 Dec 1980	42	YEARS	M	N	Surgically sterile		NOT HISPANIC ...	0	0	
10	002	002-1002	SCRN	09 Aug 1992	31	YEARS	F	Y			HISPANIC OR L...	0	0	
11	002	002-1003	SCRN	02 Jul 1990	33	YEARS	F	Y			NOT HISPANIC ...	1	0	
12	002	002-1004	SCRN	20 Aug 1972	51	YEARS	M	NA			NOT HISPANIC ...	1	0	
13	002	002-1005	SCRN	01 Jun 1970	53	YEARS	F	N	Post-menopausal		NOT HISPANIC ...	1	0	
14	002	002-1006	SCRN	15 Mar 1975	48	YEARS	F	N	Post-menopausal		NOT HISPANIC ...	0	0	
15	002	002-1007	SCRN	04 Oct 1988	35	YEARS	F	Y			NOT HISPANIC ...	0	1	Samoan
16	002	002-1008	SCRN	15 Jan 1985	38	YEARS	F	Y			NOT HISPANIC ...	1	0	
17	002	002-1009	SCRN	05 Mar 1990	33	YEARS	M	NA			NOT HISPANIC ...	0	0	
18	003	003-1001	SCRN	11 Apr 1993	30	YEARS	F	Y			NOT REPORTED	0	1	Samoan
19	003	003-1002	SCRN	30 Apr 1980	43	YEARS	M	NA			NOT REPORTED	0	0	
20	003	003-1003	SCRN	07 Feb 1977	46	YEARS	F	N	Post-menopausal		NOT HISPANIC ...	0	0	
21	003	003-1004	SCRN	12 Dec 1980	42	YEARS	M	NA			NOT HISPANIC ...	1	0	
22	003	003-1005	SCRN	30 Dec 1980	42	YEARS	M	NA			HISPANIC OR L...	0	0	
23	003	003-1006	SCRN	15 Jan 1985	38	YEARS	F	Y			NOT HISPANIC ...	1	0	
24	003	003-1007	SCRN	30 Jun 1965	58	YEARS	M	NA			NOT HISPANIC ...	1	0	
25	003	003-1008	SCRN	07 Jan 1970	53	YEARS	F	N	Post-menopausal		NOT HISPANIC ...	1	0	

Display 2. Portion of the Generated DM Dataset

CHALLENGES AND INSIGHTS IN GENERATING SYNTHETIC DATA

The LLM can generate synthetic data efficiently and effectively; however, there are challenges due to the complexity of clinical data and the limitations of the current LLM models.

LIMITATIONS OF CONTEXT LENGTH IN LLM

All modern transformer-based LLMs have context window limitations. The GPT-4o Mini model used in this project features a context window of 128,000 tokens, enabling it to process extensive input data. However, the maximum output per request is capped at 16,384 tokens. If you exceed the token limit, the model will truncate the output to fit within the allowed number of tokens, which may result in incomplete output data.

To address this issue, we generate the data in batches. For example, if we need to generate 100 rows of records, we can split the process into 10 batches, with each batch generating only 10 records. For forms with a large number of fields, we can divide the process into even smaller batches as needed. Below is the code to implement this approach:

```
def gen_records(topic, num_records, records_per_batch=10):
    all_records = pd.DataFrame()
    print('Generating:', num_records, 'records', 'for', topic, '...')
    for _ in range(num_records // records_per_batch):
      records=gen_data(Topic=topic, Nrecords=records_per_batch)
      all_records=pd.concat([all_records, records])
      print('Generated:', len(all_records), 'records', 'for', topic)
   # Generate the remaining records if num_records is not a multiple of records_per_batch
    remaining_records = num_records -len(all_records)

    if remaining_records > 0:
       records=gen_data(Topic=topic, Nrecords=remaining_records)
       all_records=pd.concat([all_records, records])
       print('Generated:', len(all_records), 'records', 'for', topic)
    return all_records
```

Program 3. Function to Generate Data in Batch

LLM MODEL HALLUCINATIONS

LLM hallucination refers to the phenomenon where a Large Language Model (LLM) generates outputs that are factually incorrect, nonsensical, or not aligned with the requirements provided. While this can be useful for generating data unrelated to real trials, we aim to make our synthetic data more realistic.

One approach is to set a low temperature value, which reduces the likelihood of the model producing wild or imaginative responses by favoring the most probable tokens. Another way to minimize hallucinations is to use explicit and clear descriptions of the fields. Refer to the prompt example for DM above for guidance.

LLM MODEL DETERMINISTIC

Being able to reproduce the same data every time the model is run is useful, especially for programming testing. There are several ways to improve reproducibility. One easy approach is to set the temperature to 0 (or close to 0). This causes the model to select the most probable next token, reducing diversity and ensuring consistent output. You can refer to the OpenAI API cookbook for more possible solutions.

It's worth mentioning that OpenAI has added a new hyperparameter (seed) to the API to improve output reproducibility. However, it is still in beta version and not stable at the moment. It is expected to improve in the future if the seed model is incorporated into the model inference process itself.

DATA CONSISTENCY ACROSS THE WHOLE STUDY

Maintaining data consistency is a challenge in real clinical trials, and it becomes even more difficult with synthetic data. To resolve discrepancies between different forms and visits, we introduce an external file called (matrix.csv), which is built based on the study design and SOA (Scheduled Activities) in the study protocol. The matrix.csv file pre-populates all records for forms, including subject IDs and the timing information for all milestones. It also includes the IDs needed to link records between forms and visits. Below is a portion of the contents from the matrix.csv file for DM

FORM	SITE	PATIENT_NAME	FOLDER	VISDAT	IC_DSSTDAT	LAST_VISDAT	SPID	RECORD_POSITION	PAGEREPEATNUMBER
DM	001	001-1001	SCRN	17-Aug-20	10-Aug-20	14OCT2021		0	0
DM	001	001-1002	SCRN	23-Nov-20	21-Nov-20	12FEB2024		0	0
DM	001	001-1003	SCRN	05-Jan-21	29-Dec-20	03JAN2022		0	0
DM	001	001-1004	SCRN	02-Jul-20	28-Jun-20	18JAN2024		0	0
DM	001	001-1005	SCRN	23-Jun-20	17-Jun-20	08FEB2021		0	0
DM	001	001-1006	SCRN	24-Dec-20	18-Dec-20	03DEC2022		0	0
DM	001	001-1007	SCRN	11-May-20	06-May-20	28NOV2023		0	0
DM	001	001-1008	SCRN	11-Dec-20	10-Dec-20	17JUN2023		0	0
DM	002	002-1001	SCRN	13-Jul-20	12-Jul-20	18FEB2021		0	0
DM	002	002-1002	SCRN	14-Nov-20	10-Nov-20	23MAR2022		0	0
DM	002	002-1003	SCRN	30-Nov-20	23-Nov-20	12MAY2021		0	0
DM	002	002-1004	SCRN	28-Jul-20	27-Jul-20	30JUN2023		0	0
DM	002	002-1005	SCRN	10-Apr-20	04-Apr-20	24FEB2023		0	0
DM	002	002-1006	SCRN	18-Feb-21	16-Feb-21	17APR2021		0	0
DM	002	002-1007	SCRN	12-Jan-21	07-Jan-21	02APR2021		0	0
DM	002	002-1008	SCRN	15-Jan-21	11-Jan-21	08MAR2021		0	0
DM	002	002-1009	SCRN	04-May-20	02-May-20	06FEB2023		0	0
DM	003	003-1001	SCRN	19-Oct-20	14-Oct-20	09JUN2021		0	0
DM	003	003-1002	SCRN	07-Jan-21	04-Jan-21	28MAY2021		0	0
DM	003	003-1003	SCRN	15-Mar-20	08-Mar-20	01JAN2023		0	0
DM	003	003-1004	SCRN	13-Dec-20	09-Dec-20	12NOV2023		0	0
DM	003	003-1005	SCRN	26-Nov-20	25-Nov-20	25SEP2021		0	0

Display 3. Portion of the Matrix.csv Contents

Below is the code to use the matrix.csv to output datasets:

```
# Load the CSV data into a DataFrame
matrix_df = pd.read_csv('./resources/matrix.csv', dtype={'SITE': str}, low_memory=False,
encoding='ISO-8859-1')
form_df = matrix_df[matrix_df['FORM'] == 'DM']
# Loop through 'SITE', 'PATIENT_NAME', and 'FOLDER' columns
dfs = []
for index, row in form_df.iterrows():
    site = row['SITE']
    patient_name = row['PATIENT_NAME']
    folder = row['FOLDER']
    record_position= row['RECORD_POSITION']
    page_repeatnumber= row['PAGEREPEATNUMBER']
    rand_df = df.sample(n=1)
    rand_df.insert(0, 'Site', site)
    rand_df.insert(1, 'Patient_Name', patient_name)
    rand_df.insert(2, 'Folder', folder)
    rand_df.insert(3, 'Record_position', record_position)
    rand_df.insert(4, 'Page_repeatnumber', page_repeatnumber)
    dfs.append(rand_df)
result = pd.concat(dfs, ignore_index=True)
print(result)
result.to_csv("./outputs/csv/dm.csv", sep="|", index=False)
pyreadstat.write_xport(result, "./outputs/sas/dm.xpt", table_name="DM")
```

Program 4. Code to Use Matrix to Output Datasets

In the code above, the external matrix.csv file is read in and used to populate several common IDs for each form. A row is randomly selected from the generated synthetic dataframe to construct a complete record in the final dataset. The final dataset is then saved as a CSV file and exported as a SAS transport file (XPT)

CONCLUSION

The integration of AI tools like OpenAI with Python libraries such as Pydantic and Pandas offers a robust framework for generating synthetic clinical trial data. This method not only enhances efficiency but also ensures high-quality data for programming testing, data simulation, and training. By automating this process, the clinical statistical programming team can use it to test programming, while statisticians can use it to generate the final mock-up sheets for reports. Ultimately, this approach shortens programming and analysis time for submission.

Looking ahead, generating a large amount of synthetic data can consume many tokens, potentially incurring significant costs. To mitigate this, you can use OLIMA or AI Studio to deploy local, free LLM models such as Meta LLaMA or Mistral, among others. Additionally, fine-tuning an LLM with your own data can help generate more realistic, study-specific data.

REFERENCES

OpenAI API Documentation: https://platform.openai.com/docs/

Pydantic Documentation: https://pydantic-docs.helpmanual.io/

Pandas Documentation: https://pandas.pydata.org/

ACKNOWLEDGMENTS

I'd like to thank Emily Ma for testing the Python code and GitHub repository deployment.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Max Ma
Everest Clinical Research Corporation
max.ma@ecrscorp.com
https://www.ecrscorp.com/

APPENDIX COMPLETE PROGRAM LISTING

dm.py:

```
#*******************************************************************************************
#Project ID::                PharmaSUG-2025
#Program Name:              dm.py
#Original Author:           Max Ma
#Date Initiated:            20JAN25
#*******************************************************************************************

import os
from dotenv import load_dotenv
import instructor
from pydantic import BaseModel, Field
from openai import OpenAI, OpenAIError
from typing import List
import pandas as pd
import pyreadstat
import random

class Metadata(BaseModel):

    BRTHDAT: str = Field(..., description="Date of Birth, in format capital dd MMM yyyy")
    AGE: int = Field(...,  description="Age between 18 and 80, a numeric value in the
format 3,  derived from BRTHDAT based on today's date.")
    AGEU: str = Field(..., description="Age Units, Text coded with AGEU, the value must
be in ('DAYS', 'HOURS', 'MONTHS', 'WEEKS', 'YEARS'), 'DAYS' has a 0.0% chance, 'HOURS'
has a 0.0% chance, 'MONTHS' has a 0.0% chance, 'WEEKS' has a 0.0% chance, 'YEARS' has a
100.0% chance")
    SEX: str = Field(..., description="Sex, Text coded with SEX1, the value must be in
('F', 'M'), 'F' has a 50.0% chance, 'M' has a 50.0% chance")
    CHILDPOT: str = Field(..., description="Childbearing Potential, the value must be in
('Y', 'N') for Female, 'NA' for male.")
    CHILDPOTRSN: str = Field(..., description="Reason not childbearing, must be generated
for CHILDPOT=N only, the value must be in ('Post-menopausal', 'Surgically sterile',
'Other') for female,'Post-menopausal' should be generated for female between the ages of
45 and 55, blank for male")
    CHILDPOTSP: str = Field(..., description="If Other Specify, generate it only the
corresponding item is 'Other', free text up to 200 characters long.")
    ETHNIC: str = Field(..., description="Ethnicity, Text coded with ETHNIC, the value
must be in ('HISPANIC OR LATINO', 'NOT HISPANIC OR LATINO', 'NOT REPORTED', 'UNKNOWN'),
'HISPANIC OR LATINO' has a 2.0% chance, 'NOT HISPANIC OR LATINO' has a 97.0% chance, 'NOT
REPORTED' has a 0.5% chance, 'UNKNOWN' has a 0.5% chance")
    RACE_WHITE: str = Field(..., pattern=r"^(0|1)$", description="Race (White), a numeric
value in the format 1, the value must be in ('1 = Checked', '0 = Not Checked'), '1' has a
40.0% chance")
    RACE_ASIAN: str = Field(..., pattern=r"^(0|1)$", description="Race (Asian), a numeric
value in the format 1, the value must be in ('1 = Checked', '0 = Not Checked'), '1' has a
10.0% chance")
    RACE_BLACK: str = Field(..., pattern=r"^(0|1)$", description="Race (Black or African
American), a numeric value in the format 1, the value must be in ('1 = Checked', '0 = Not
Checked'), '1' has a 10.0% chance")
    RACE_AMERIND: str = Field(..., pattern=r"^(0|1)$", description="Race (American Indian
or Alaska Native), a numeric value in the format 1, the value must be in ('1 = Checked',
'0 = Not Checked'), '1' has a 10.0% chance")
    RACE_NHAWII: str = Field(..., pattern=r"^(0|1)$", description="Race (Native Hawaiian
Pacific Islanders), a numeric value in the format 1, the value must be in ('1 = Checked',
'0 = Not Checked'), '1' has a 10.0% chance")
    RACE_NR: str = Field(..., pattern=r"^(0|1)$", description="Race (Not Reported), a
numeric value in the format 1, the value must be in ('1 = Checked', '0 = Not Checked'),
'1' has a 10.0% chance")
    RACE_OTHER: str = Field(..., pattern=r"^(0|1)$", description="Race (Other), a numeric
value in the format 1, the value must be in ('1 = Checked', '0 = Not Checked'), '1' has a
10.0% chance")
    RACEOTH: str = Field(..., description="If Other Specify, must generate it if the
field RACE_OTHER is '1', free text up to 200 characters long, for example: 'Caribbean',
```

```
'Aboriginal Australian', 'Samoan' or 'Tongan', etc., if it's null and RACE_OTHER='1', set
it to 'Unknown")

class OutList(BaseModel):
    outlist: List[Metadata] = Field(..., description="Multiple records")

# Patch the OpenAI client
load_dotenv()
client = instructor.from_openai(OpenAI(api_key=os.environ.get("OPENAI_API_KEY")))

# Extract structured data from natural language
def gen_data(Topic: str, Nrecords: str) -> OutList:
    prompt = f"Generate exactly {Nrecords} {Topic} records."

    try:
      # Make your OpenAI API request here
      #response = client.chat.completions.create_iterable(
      response = client.chat.completions.create(
        model="gpt-4o-mini",
        max_retries=3, # Retry the request 3 times
        response_model=OutList,
        messages=[
           {"role": "system", "content": "You are a clinical expert specializing in the
generation of high-quality synthetic data for clinical trial. Your task is to create
synthetic data that strictly conforms to the provided specifications for each field,
ensuring realism and logical consistency. The generated data must be accurate, plausible,
and appropriate for rigorous testing and validation purposes."},
           {"role": "user", "content": prompt}
        ],
        temperature=0.7,
        #top_p=0.1,
        #seed=8
      )
    except OpenAIError as e:
      # Handle all OpenAI API errors
      print(f'Error: {e}')

    records=vars(response).get('outlist')
    df=pd.DataFrame([vars(s) for s in records])
    #print(df)
    return df

def gen_records(topic, num_records, records_per_batch=10):
    all_records = pd.DataFrame()
    print('Generating:', num_records, 'records', 'for', topic, '...')
    for _ in range(num_records // records_per_batch):
      records=gen_data(Topic=topic, Nrecords=records_per_batch)
      all_records=pd.concat([all_records, records])
      print('Generated:', len(all_records), 'records', 'for', topic)

    # Generate the remaining records if num_records is not a multiple of
records_per_batch
    remaining_records = num_records -len(all_records)
    if remaining_records > 0:
       records=gen_data(Topic=topic, Nrecords=remaining_records)
       all_records=pd.concat([all_records, records])
       print('Generated:', len(all_records), 'records', 'for', topic)
    return all_records

print('Generating:', 'DM', '...')
df=gen_records(topic='Demographics', num_records=100, records_per_batch=10)

# Load the CSV data into a DataFrame
matrix_df = pd.read_csv('./resources/matrix.csv', dtype={'SITE': str}, low_memory=False,
encoding='ISO-8859-1')
form_df = matrix_df[matrix_df['FORM'] == 'DM']
# Loop through 'SITE', 'PATIENT_NAME', and 'FOLDER' columns
```

```
dfs = []
for index, row in form_df.iterrows():
    site = row['SITE']
    patient_name = row['PATIENT_NAME']
    folder = row['FOLDER']
    record_position= row['RECORD_POSITION']
    page_repeatnumber= row['PAGEREPEATNUMBER']
    rand_df = df.sample(n=1)
    rand_df.insert(0, 'Site', site)
    rand_df.insert(1, 'Patient_Name', patient_name)
    rand_df.insert(2, 'Folder', folder)
    rand_df.insert(3, 'Record_position', record_position)
    rand_df.insert(4, 'Page_repeatnumber', page_repeatnumber)
    dfs.append(rand_df)

result = pd.concat(dfs, ignore_index=True)
print(result)
result.to_csv("./outputs/csv/dm.csv", sep="|", index=False)
pyreadstat.write_xport(result, "./outputs/sas/dm.xpt", table_name="DM")
```

ae.py:

```
#*********************************************************************************************
#Project ID::               PharmaSUG-2025
#Program Name:              ae.py
#Original Author:           Max Ma
#Date Initiated:            20JAN25
#*********************************************************************************************

import os
from dotenv import load_dotenv
import instructor
from pydantic import BaseModel, Field
from openai import OpenAI, OpenAIError
from typing import List
import pandas as pd
import pyreadstat
import random
from datetime import datetime, timedelta

class Metadata(BaseModel):

    AESPID: str = Field(..., description="AE Number, free text up to 4 characters long.")
    AETERM: str = Field(..., description="Reported Term for the Adverse Event, free text
up to 200 characters long.")
    AESTDAT: str = Field(..., description="Adverse Event Start Date, in format capital dd
MMM yyyy.")
    AESTTIM: str = Field(..., description="Start Time of Adverse Event, in format
HH:nn.")
    AESTTIMUNK: str = Field(..., description="Start Time of Adverse Event Unknown, set to
blank if it's ongoing, set to 0 if the corresponding time is not missing, otherwise set
it to 1, a numeric value in the format 1, the value must be in ('1 = Checked', '0 = Not
Checked')")
    AEONGO: str = Field(..., description="Ongoing Adverse Event, Text coded with
AEONGO_NY1, the value must be in ('Y', 'N'), 'Y' has a 18.0% chance, 'N' has a 82.0%
chance")
    AEENDAT: str = Field(..., description="Adverse Event End Date, it shoul be after the
start date or time , set to blank if it's ongoing, in format capital dd MMM yyyy.")
    AEENTIM: str = Field(..., description="End Time of Adverse Event, it shoul be after
the start date or time , set to blank if it's ongoing, in format HH:nn.")
    AEENTIMUNK: str = Field(..., description="End Time of Adverse Event Unknown, set to
blank if it's ongoing, set to 0 if the corresponding time is not missing, otherwise set
it to 1, a numeric value in the format 1, the value must be in ('1 = Checked', '0 = Not
Checked')")
    AEOUT: str = Field(..., description="Outcome of Adverse Event, Text coded with OUT,
the value must be in ('RECOVERED/RESOLVED', 'RECOVERED/RESOLVED WITH SEQUELAE',
```

```
'RECOVERING/RESOLVING', 'NOT RECOVERED/NOT RESOLVED', 'FATAL', 'UNKNOWN'),
'RECOVERED/RESOLVED' has a 50.0% chance, 'RECOVERED/RESOLVED WITH SEQUELAE' has a 10.0%
chance, 'RECOVERING/RESOLVING' has a 20.% chance, 'NOT RECOVERED/NOT RESOLVED' has a
10.0% chance, 'FATAL' has a 5.0% chance, 'UNKNOWN' has a 5.0% chance")
    AESEV: str = Field(..., description="AE Severity/Intensity, Text coded with AESEV,
the value must be in ('MILD', 'MODERATE', 'SEVERE'), 'MILD' has a 65.0% chance,
'MODERATE' has a 26.0% chance, 'SEVERE' has a 9.0% chance")
    AESCALE: str = Field(..., description="AE Grading Scale, Text coded with AESCALE, the
value must be in ('NCI CTC AE v5.0', 'CBER Vaccine Grading'), 'NCI CTC AE v5.0' has a
100.0% chance, 'CBER Vaccine Grading' has a 0.0% chance")
    AETOXGR: str = Field(..., description="AE Standard Toxicity Grade, Text coded with
AETOXGR, the value must be in ('1', '2', '3', '4', '5'), must be '5' where AEOUT is
'FATAL', '1' has a 65.0% chance, '2' has a 26.0% chance, '3' has a 8.0% chance, '4' has a
0.5% chance, '5' has a 0.5% chance")
    AEREL: str = Field(..., description="AE Causality Primary, Text coded with AEREL, the
value must be in ('NOT RELATED', 'UNLIKELY RELATED', 'POSSIBLY RELATED', 'RELATED'), 'NOT
RELATED' has a 10.0% chance, 'UNLIKELY RELATED' has a 10.0% chance, 'POSSIBLY RELATED'
has a 10.0% chance, 'RELATED' has a 70.0% chance")
    AEACN: str = Field(..., description="Action Taken with Primary Study Treatment, Text
coded with ACN, the value must be in ('DOSE INCREASED', 'DOSE NOT CHANGED', 'DOSE RATE
REDUCED', 'DOSE REDUCED', 'DRUG INTERRUPTED', 'DRUG WITHDRAWN', 'NOT APPLICABLE',
'UNKNOWN')")
    AECONTRT: str = Field(..., description="Concomitant or Additional Trtmnt Given, Text
coded with AECONTRT_NY1, the value must be in ('Y', 'N'), 'Y' has a 36.0% chance, 'N' has
a 64.0% chance")
    AEACNOTH: str = Field(..., pattern="^(0|1)$", description="Other Action Taken -
Other, a numeric value in the format 1, the value must be in ('1 = Checked', '0 = Not
Checked'), '1' has a 20.0% chance")
    AEACNOTHSP: str = Field(..., description="Other Action Taken Specify, generate it
only the corresponding item is 'Other', free text up to 200 characters long.")
    AESER: str = Field(..., description="AE Serious Event, Text coded with AESER_NY1, the
value must be in ('Y', 'N'), 'Y' has a 6.0% chance, 'N' has a 94.0% chance")
    AESI: str = Field(..., description="Adverse Event of Special Interest, Text coded
with AESI_NY1, the value must be in ('Y', 'N'), 'Y' has a 2.5% chance, 'N' has a 97.5%
chance")
    AEDLT: str = Field(..., description="Dose Limiting Toxicity, Text coded with
AEDLT_NY1, the value must be in ('Y', 'N'), 'Y' has a 0.0% chance, 'N' has a 100.0%
chance")
    AEMED: str = Field(..., description="AE Medically Attended, Text coded with
AEMED_NY1, the value must be in ('Y', 'N'), 'Y' has a 25.0% chance, 'N' has a 75.0%
chance")

class OutList(BaseModel):
    outlist: List[Metadata] = Field(..., description="Multiple records")

# Patch the OpenAI client
load_dotenv()
client = instructor.from_openai(OpenAI(api_key=os.environ.get("OPENAI_API_KEY")))

# Extract structured data from natural language
def gen_data(Topic: str, Nrecords: str) -> OutList:
    prompt = f"Generate exactly {Nrecords} {Topic} records."

    try:
      # Make your OpenAI API request here
      #response = client.chat.completions.create_iterable(
      response = client.chat.completions.create(
        model="gpt-4o-mini",
        max_retries=3, # Retry the request 3 times
        response_model=OutList,
        messages=[
           {"role": "system", "content": "You are a clinical expert specializing in the
generation of high-quality synthetic data for clinical trial. Your task is to create
synthetic data that strictly conforms to the provided specifications for each field,
ensuring realism and logical consistency. The generated data must be accurate, plausible,
and appropriate for rigorous testing and validation purposes."},
           {"role": "user", "content": prompt}
```

```
        ],
        temperature=0.6,
        #top_p=0.1,
        seed=8
      )
    except OpenAIError as e:
      # Handle all OpenAI API errors
      print(f'Error: {e}')

    records=vars(response).get('outlist')
    df=pd.DataFrame([vars(s) for s in records])
    print(df)
    return df

def gen_records(topic, num_records, records_per_batch=10):
    all_records = pd.DataFrame()
    print('Generating:', num_records, 'records', 'for', topic, '...')
    for _ in range(num_records // records_per_batch):
      records=gen_data(Topic=topic, Nrecords=records_per_batch)
      all_records=pd.concat([all_records, records])
      print('Generated:', len(all_records), 'records', 'for', topic)

    # Generate the remaining records if num_records is not a multiple of
records_per_batch
    remaining_records = num_records -len(all_records)
    if remaining_records > 0:
       records=gen_data(Topic=topic, Nrecords=remaining_records)
       all_records=pd.concat([all_records, records])
       print('Generated:', len(all_records), 'records', 'for', topic)
    return all_records

print('Generating:', 'AE', '...')
df=gen_records(topic='Adverse Events', num_records=100, records_per_batch=10)

# Load the CSV data into a DataFrame
matrix_df = pd.read_csv('./resources/matrix.csv', dtype=str, low_memory=False,
encoding='ISO-8859-1')
form_df = matrix_df[matrix_df['FORM'] == 'AE']
master_df = pd.read_csv('./resources/aeterms.csv')
# Loop through 'SITE', 'PATIENT_NAME', and 'FOLDER' columns
dfs = []
for index, row in form_df.iterrows():
    site = row['SITE']
    patient_name = row['PATIENT_NAME']
    folder = row['FOLDER']
    record_position= row['RECORD_POSITION']
    page_repeatnumber= row['PAGEREPEATNUMBER']
    ic_dsstdat= datetime.strptime(row['IC_DSSTDAT'].replace('-', ''), '%d%b%Y')
    max_visdatn= datetime.strptime(row['LAST_VISDAT'].replace('-', ''), '%d%b%Y')
    rand_df = df.sample(n=1)
    rand_df.insert(0, 'Site', site)
    rand_df.insert(1, 'Patient_Name', patient_name)
    rand_df.insert(2, 'Folder', folder)
    rand_df.insert(3, 'Record_position', record_position)
    rand_df.insert(4, 'Page_repeatnumber', page_repeatnumber)
    if rand_df['AESPID'].iloc[0] != '':
       rand_df['AESPID'] = row['SPID']
    rand_df['AETERM'] = master_df.sample(n=1)['AETERM'].values[0]
    if rand_df['AESTDAT'].iloc[0] != '':
       rand_df['AESTDAT'] = (ic_dsstdat + timedelta(days=random.randint(0,
abs((max_visdatn - ic_dsstdat).days)))).strftime('%d %b %Y').upper()
    if rand_df['AEENDAT'].iloc[0] != '':
       rand_df['AEENDAT'] = (datetime.strptime(rand_df['AESTDAT'].iloc[0], '%d %b %Y') +
timedelta(days=random.randint(0, 30))).strftime('%d %b %Y').upper()
    dfs.append(rand_df)

result = pd.concat(dfs, ignore_index=True)
```

```
print(result)
result.to_csv("./outputs/csv/ae.csv", sep="|", index=False)
pyreadstat.write_xport(result, "./outputs/sas/ae.xpt", table_name="AE")
```

vs.py:

```
#*******************************************************************************
#Project ID::               PharmaSUG-2025
#Program Name:              vs.py
#Original Author:           Max Ma
#Date Initiated:            20JAN25
#*******************************************************************************

import os
from dotenv import load_dotenv
import instructor
from pydantic import BaseModel, Field
from openai import OpenAI, OpenAIError
from typing import List
import pandas as pd
import pyreadstat
import random
from datetime import datetime, timedelta

class Metadata(BaseModel):

    VSPERF: str = Field(..., description="Vital Signs Performed, Text coded with NY1, the
value must be in ('Y', 'N')")
    VSPERF_REASND: str = Field(..., description="Vital Signs Reason Not Per at Visit,
generate only if the abvoe answer is 'No', Text coded with REASND, the value must be in
('Adverse Event', 'Missed in Error', 'Physician Decision', 'Other'), 'Adverse Event' has
a 50.0% chance, 'Missed in Error' has a 10.0% chance, 'Physician Decision' has a 30.0%
chance, 'Other' has a 10.0% chance")
    VSPERF_REASNDSP: str = Field(..., description="Vital Signs Reason Not Per at Visit
Spec, generate only if the above reason is 'Other', free text up to 200 characters
long.")
    VSDAT: str = Field(..., description="Vital Signs Date, in format capital dd MMM
yyyy.")
    VSTIM: str = Field(..., description="Vital Signs Time, in format HH:nn.")
    VSTIMUNK: str = Field(..., description="Vital Signs Time Unknown, set to blank if
it's ongoing, set to 0 if the corresponding time is not missing, otherwise set it to 1, a
numeric value in the format 1, the value must be in ('1 = Checked', '0 = Not Checked')")
    TEMP_ORRES: str = Field(..., description="Temperature Result in degrees Celsius, a
numeric value in the format 4.1.")
    TEMP_LOC: str = Field(..., description="Temperature Anatomical Location, Text coded
with LOC1, the value must be in ('ORAL CAVITY', 'AXILLA', 'TYMPANIC MEMBRANE', 'OTHER'),
'ORAL CAVITY' has a 56.6% chance, 'AXILLA' has a 0.5% chance, 'TYMPANIC MEMBRANE' has a
20.4% chance, 'OTHER' has a 22.5% chance")
    TEMP_LOCSP: str = Field(..., description="Temperature Anatomical Location Specify,
free text up to 200 characters long.")
    RESP_ORRES: str = Field(..., description="Respiratory Rate Result, a numeric value in
the format 2.")
    SYSBP_ORRES: str = Field(..., description="Systolic Blood Pressure Result, a numeric
value in the format 3.")
    DIABP_ORRES: str = Field(..., description="Diastolic Blood Pressure Result, a numeric
value in the format 3.")
    PULSE_ORRES: str = Field(..., description="Pulse Result, a numeric value in the
format 3.")
    OXYSAT_ORRES: str = Field(..., description="Oxygen Saturation Result, a numeric value
in the format 3.")
    VSPOS: str = Field(..., description="Vital Signs Position of Subject, Text coded with
POSITION, the value must be in ('PRONE', 'SUPINE', 'SITTING', 'STANDING'), 'PRONE' has a
0.0% chance, 'SUPINE' has a 0.0% chance, 'SITTING' has a 100.0% chance, 'STANDING' has a
0.0% chance")
```

```
    VSDESC: str = Field(..., description="Vital Signs Abnormal Findings, write one or two
sentence descriptuon about the vital signs test results based on the TEMP_ORRES,
RESP_ORRES, SYSBP_ORRES, DIABP_ORRES, PULSE_ORRES and OXYSAT_ORRES.")
    VSCLSIG: str = Field(..., description="Vital Signs Clinical Significance, generate
only if the above result is 'Abnormal', Text coded with NY1, the value must be in ('Y',
'N')")

class OutList(BaseModel):
    outlist: List[Metadata] = Field(..., description="Multiple records")

# Patch the OpenAI client
load_dotenv()
client = instructor.from_openai(OpenAI(api_key=os.environ.get("OPENAI_API_KEY")))

# Extract structured data from natural language
def gen_data(Topic: str, Nrecords: str) -> OutList:
    prompt = f"Generate exactly {Nrecords} {Topic} records."

    try:
      # Make your OpenAI API request here
      #response = client.chat.completions.create_iterable(
      response = client.chat.completions.create(
        model="gpt-4o-mini",
        max_retries=3, # Retry the request 3 times
        response_model=OutList,
        messages=[
           {"role": "system", "content": "You are a clinical expert specializing in the
generation of high-quality synthetic data for clinical trial. Your task is to create
synthetic data that strictly conforms to the provided specifications for each field,
ensuring realism and logical consistency. The generated data must be accurate, plausible,
and appropriate for rigorous testing and validation purposes."},
           {"role": "user", "content": prompt}
        ],
        temperature=0.6,
        #top_p=0.1,
        seed=8
      )
    except OpenAIError as e:
      # Handle all OpenAI API errors
      print(f'Error: {e}')

    records=vars(response).get('outlist')
    df=pd.DataFrame([vars(s) for s in records])
    print(df)
    return df

def gen_records(topic, num_records, records_per_batch=10):
    all_records = pd.DataFrame()
    print('Generating:', num_records, 'records', 'for', topic, '...')
    for _ in range(num_records // records_per_batch):
      records=gen_data(Topic=topic, Nrecords=records_per_batch)
      all_records=pd.concat([all_records, records])
      print('Generated:', len(all_records), 'records', 'for', topic)

    # Generate the remaining records if num_records is not a multiple of
records_per_batch
    remaining_records = num_records -len(all_records)
    if remaining_records > 0:
       records=gen_data(Topic=topic, Nrecords=remaining_records)
       all_records=pd.concat([all_records, records])
       print('Generated:', len(all_records), 'records', 'for', topic)
    return all_records

print('Generating:', 'VS', '...')
df=gen_records(topic='Vital Signs', num_records=100, records_per_batch=10)

# Load the CSV data into a DataFrame
```

```
matrix_df = pd.read_csv('./resources/matrix.csv', dtype={'SITE': str}, low_memory=False,
encoding='ISO-8859-1')
matrix_df.loc[matrix_df['VISDAT'].isna(), 'VISDAT'] = matrix_df['LAST_VISDAT']
form_df = matrix_df[matrix_df['FORM'] == 'VS']
# Loop through 'SITE', 'PATIENT_NAME', and 'FOLDER' columns
dfs = []
for index, row in form_df.iterrows():
    site = row['SITE']
    patient_name = row['PATIENT_NAME']
    folder = row['FOLDER']
    record_position= row['RECORD_POSITION']
    page_repeatnumber= row['PAGEREPEATNUMBER']
    visdat= datetime.strptime(row['VISDAT'].replace('-', ''), '%d%b%Y')
    rand_df = df.sample(n=1)
    rand_df.insert(0, 'Site', site)
    rand_df.insert(1, 'Patient_Name', patient_name)
    rand_df.insert(2, 'Folder', folder)
    rand_df.insert(3, 'Record_position', record_position)
    rand_df.insert(4, 'Page_repeatnumber', page_repeatnumber)
    if rand_df['VSDAT'].iloc[0] != '':
       rand_df['VSDAT'] = visdat.strftime('%d %b %Y').upper()
    dfs.append(rand_df)

result = pd.concat(dfs, ignore_index=True)
print(result)
result.to_csv("./outputs/csv/vs.csv", sep="|", index=False)
pyreadstat.write_xport(result, "./outputs/sas/vs.xpt", table_name="VS")
```