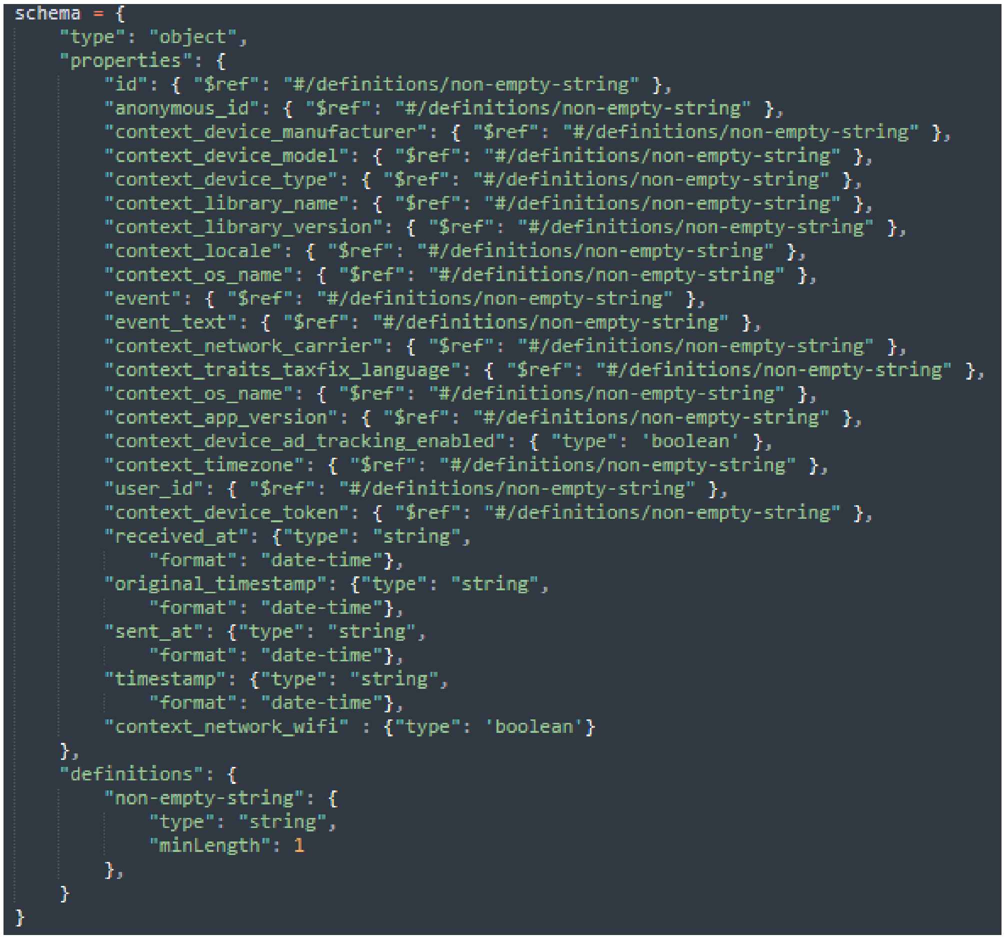
# JSON Schema Validator

## Part 1: Understanding the data in input.json

| Sr | Json field | Field type |
|---|---|---|
| 1 | id | non-empty string |
| 2 | received_at | date-time |
| 3 | anonymous_id | non-empty string |
| 4 | context_device_manufacturer | non-empty string |
| 5 | context_device_model | non-empty string |
| 6 | context_device_type | non-empty string |

| | | |
|---|---|---|
| 7 | context_library_name | non-empty string |
| 8 | context_library_version | non-empty string |
| 9 | context_locale | non-empty string |
| 10 | context_network_wifi | boolean[true/false] |
| 11 | context_os_name | non-empty string |
| 12 | event | non-empty string |
| 13 | event_text | non-empty string |
| 14 | original_timestamp | DateTime |
| 15 | sent_at | DateTime |
| 16 | timestamp | DateTime |
| 17 | context_network_carrier | non-empty string |
| 18 | context_traits_taxfix_language | non-empty string |
| 19 | context_app_version | non-empty string |
| 20 | context_device_as_tracking_enabled | boolean[true/false] |
| 21 | context_time_zone | non-empty string |
| 22 | user_id | non-empty string |
| 23 | context_device_token | non-empty string |

## Part 2: Create a schema validator



```
schema = {
    "type": "object",
    "properties": {
        "id": { "$ref": "#/definitions/non-empty-string" },
        "anonymous_id": { "$ref": "#/definitions/non-empty-string" },
        "context_device_manufacturer": { "$ref": "#/definitions/non-empty-string" },
        "context_device_model": { "$ref": "#/definitions/non-empty-string" },
        "context_device_type": { "$ref": "#/definitions/non-empty-string" },
        "context_library_name": { "$ref": "#/definitions/non-empty-string" },
        "context_library_version": { "$ref": "#/definitions/non-empty-string" },
        "context_locale": { "$ref": "#/definitions/non-empty-string" },
        "context_os_name": { "$ref": "#/definitions/non-empty-string" },
        "event": { "$ref": "#/definitions/non-empty-string" },
        "event_text": { "$ref": "#/definitions/non-empty-string" },
        "context_network_carrier": { "$ref": "#/definitions/non-empty-string" },
        "context_traits_taxfix_language": { "$ref": "#/definitions/non-empty-string" },
        "context_os_name": { "$ref": "#/definitions/non-empty-string" },
        "context_app_version": { "$ref": "#/definitions/non-empty-string" },
        "context_device_ad_tracking_enabled": { "type": 'boolean' },
        "context_timezone": { "$ref": "#/definitions/non-empty-string" },
        "user_id": { "$ref": "#/definitions/non-empty-string" },
        "context_device_token": { "$ref": "#/definitions/non-empty-string" },
        "received_at": {"type": "string",
            "format": "date-time"},
        "original_timestamp": {"type": "string",
            "format": "date-time"},
        "sent_at": {"type": "string",
            "format": "date-time"},
        "timestamp": {"type": "string",
            "format": "date-time"},
        "context_network_wifi" : {"type": 'boolean'}
    },
    "definitions": {
        "non-empty-string": {
            "type": "string",
            "minLength": 1
        },
    }
}
```

Fig. 1. Json schema validator definition

## Assumptions:

1. For the json data containing the errors(wrong data type or empty value), the data will be recorded as it is in the log json file in the output_json_log folder.
2. The analysis(date, event, and number of events by date) done on the json data is recorded in the CSV file in the csv_generated_report folder. Here, the analysis is performed taking the **timestamp** attribute, **event** attribute in the json.

## Python modules used:

| Module name | Purpose | Reference link |
|---|---|---|
| ijson | Iterative json parser with a standard Python iterator interface | ijson |
| jsonschema | Implementation of the JSON Schema | jsonschema |

| | | |
|---|---|---|
| | specification for Python | |
| pandas | Python package for data analysis | pandas |
| memory-profiler | monitoring memory consumption of a process as well as line-by-line analysis of memory consumption for python programs | memory-profiler |
| json | Dealing with json | json |
| datetime | Manipulating date and time | datetime |
| os | Manipulate paths | os |
| unittest | To perform unit testing | unittest |

### Why ijson?

- ijson is an iterative JSON parser with a standard Python iterator interface.
- It can process multi GB json file without having memory shortage problems.

### json vs ijson:

- json.load() read the whole JSON document into memory before parsing it.
- json.load() first reads the whole document into memory as a string. It then starts parsing that string and converting the whole document into python types again stored in memory.
- ijson does not read the whole document into memory.
- When using the ijson.items(), the prefix works as the selection for which objects should be automatically built and returned by ijson.
- Ijson provides several implementations of the actual parsing in the form of backends located in ijson/backends. (yajl2, yajl, python, etc.)

### Related links:

1. yajl
2. understanding json schema

**<u>Work Flow</u>**:

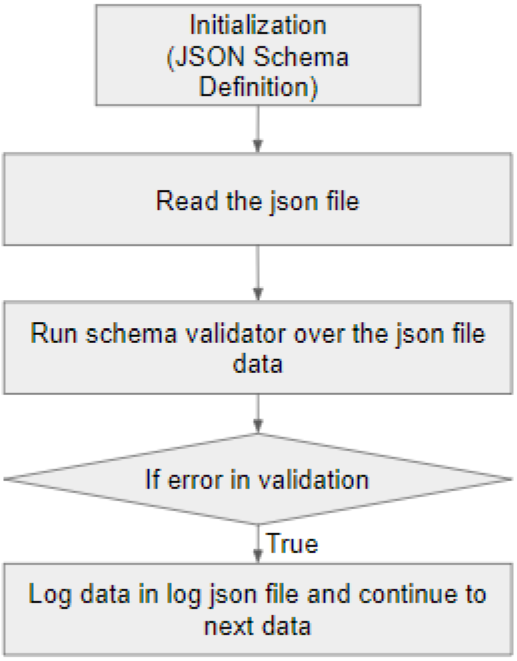

Initialization (JSON Schema Definition)
Read the json file
Run schema validator over the json file data
If error in validation
True
Log data in log json file and continue to next data

Fig. 2. WorkFlow for JSON Schema Validation

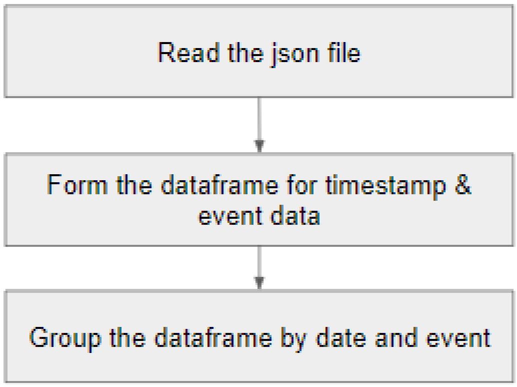
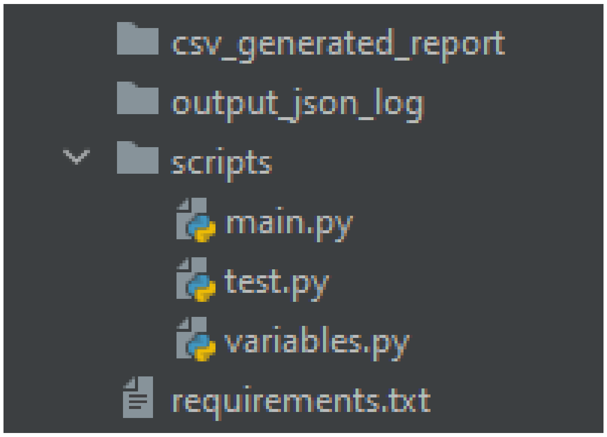

Read the json file
Form the dataframe for timestamp & event data
Group the dataframe by date and event

Fig. 3. WorkFlow for generating date and event count-based report

## Part 3: Project Folder Structure & Environment Details

### Project Folder Structure:



```
    csv_generated_report
    output_json_log
  ˅ scripts
        main.py
        test.py
        variables.py
    requirements.txt
```

Fig. 4. Project Folder Structure

### Environment Details:

Operating System: Windows 10 64 bit

Programming Language: Python 3.8.6

Text Editor to write the code: Sublime Text Editor

## Part 4: How to reproduce the project

### Cloning the project from GitHub:

**Step 1:** Using the Git CLI, copy the following command:

```
git clone https://github.com/itsvaishnavi/JSON_Schema_Validator.git
```

To download git, visit https://git-scm.com/downloads

**Step 2:** Navigate to the folder where the project folder is cloned. Open the command prompt at this path and run the following command:

```
pip install -r requirements.txt
```

This downloads all the essential python modules required for the project.

**Step 3:** We are now ready to run the python scripts. There are three Python scripts.

| File Name | Parent folder | Comments |
|---|---|---|
| main.py | scripts | Script to implement the JSON schema validator & validate the JSON. |
| variables.py | scripts | Script where the JSON schema is defined based on input.json |
| test.py | scripts | Script contains the unit test cases. |

To implement the JSON schema validator & validate the JSON, navigate to the **scripts** folder and run the following command to run main.py:

```
python main.py <fileName.json>
```

To run the test cases, navigate to the **scripts** folder and run the following command to run test.py:

```
python -m unittest
```

**Author**

**Vaishnavi Piyush Kand**