CDISC CORE AND 'HOMEMADE CORE': ENHANCING DATA CHECK OPERATIONS

June 28, 2023
AGENDA

CORE

RULES

Homemade CORE
CDISC CORE is a project that aims to deliver clear and enforceable Conformance Rules for each Foundational Standard. It provides an open-source execution engine as a Reference Implementation for these rules.
The CORE Project objectives are to:

- Ensure each standard has a set of unambiguous, executable Conformance Rules
- Ensure consistency across Conformance Rule implementations
- Expedite the availability of executable Conformance Rules for new Foundational Standards
- Create executable Conformance Rules vetted by the CDISC standards development teams
- Create a Reference Implementation of an open-source engine that executes the Rules
- Release the open-source engine under the CDISC Open-Source Alliance (COSA)
The following diagram illustrates the concept of the CORE project, including the Conformance Rules, the executable form of the Rules, and the Rules execution engine:
CORE Program Roadmap

Evaluation Release
- **Engine**: Open-Source, developed by CDISC, published under COSA
- **Conformance Rules**: SDTM 2.0 and SDTMIG 3.4
- **Functionality**: Basic conformance checking functionality
- **Deployments**: CDISC cloud evaluation
- **Deployments**: Azure Marketplace evaluation
- **Deployments**: Desktop evaluation – not achieved yet

Production Release 1: Submission-ready Engine and Rules
- **Engine**: Open-Source under COSA; evolved; maintained by the open-source software community
- **Conformance Rules**: Remainder of CDISC Foundational Standards
- **Functionality**: Complete conformance checking functionality
- **Deployments**: Vendor- or user-provided cloud & local production environments

Production Release 2: Enhanced Engine and Rules
- **Engine**: Open-Source under COSA, evolved, maintained by the open-source software community
- **Conformance Rules**: New CDISC Standards released with Conformance Rules
- **Functionality**: Advanced functionality
- **Deployments**: Vendor- or user-provided cloud & local production environments

Q3 2021 – Q2 2022

Q3 2022 – Q2 2023

Q3 2023 – Q2 2024

Establish CORE Roadmap Board

CDISC
Vendor/User
CORE – Engine

Functionality:

> Executes CORE Rules (YAML) against clinical data and returns results
> Deployment agnostic
> Open-source, available in GitHub

Current focus:

> Process new YAML operators added to express new rules
> Process new clinical data formats
> Support Define xml crosschecking
CORE – Is Open-Source

- Open-source framework
- Listed in the COSA (CDISC Open-Source Alliance) directory
- Permissive MIT open-source license
- Provided via GitHub
- Free to all in CDISC community
- Very flexible implementation options
CORE – In Action

CMD

FORMEDIX

CORE

Pinnacle 21
**CORE – CMD**

The CDISC Core Engine provides a command-line interface (CMD), which serves as an interface for users to interact with the software and execute a wide range of operations.

- **\CORE**
- **VALIDATE**
- **-S SDTMIG**
- **-V 3-2**
- **-D .\XPT\**

---

```plaintext
--HELP

cache TEXT  Relative path to cache files containing pre-labeled metadata and rules
-p, --pool-size INTEGER  Number of parallel processes for validation
-d, --data TEXT  Path to directory containing data files
-dm, --dataset path TEXT  Absolute path to dataset file. Can be specified multiple times.
-l, --log-level [info|debug|error|critical|disable|warn]  Sets log level for engine logs, logs are disabled by default
-rt, --report-template TEXT  File path of report template to use for excel output
-s, --standard TEXT  CDISC standard to validate against (required)
-v, --version TEXT  Standard version to validate against (required)
-ct, --controlled-terminology-package TEXT  Controlled terminology package to validate against, can provide more than one
-o, --output TEXT  Report output file destination
-of, --output-format [xsdsdom|xml]  Output file format
-rr, --raw-report  Report in a raw format as it is generated by the engine, this flag must be used only with --output-format XML
-dx, --define-version TEXT  Define XML version used for validation
-wd, --working-dir TEXT  Path to directory with MODdata dictionary files
-md, --moddata TEXT  Path to directory with MODdata dictionary files
-r, --rules TEXT  Rule core id. ex: CORE-xxxxxxx, can be specified multiple times.
-v, --verbose-output  Specify this option to print rules as they are completed
-p, --progress [verbose_output|disabled|progress|percent]  Defines how to display the validation progress. By default a progress bar like

```

78% is printed.
# CORE – CMD Results

## Conformance Details

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report Generation</td>
<td>2023-05-22T05:39:17</td>
</tr>
<tr>
<td>Total Runtime</td>
<td>63.5 seconds</td>
</tr>
<tr>
<td>CORE Engine Version</td>
<td>0.6.1</td>
</tr>
</tbody>
</table>

## Standards Details

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>SDTMIG</td>
</tr>
<tr>
<td>Version</td>
<td>V3.2</td>
</tr>
</tbody>
</table>

## Issue Details

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORE-ID</td>
<td>Message</td>
<td>Executability</td>
<td>Dataset</td>
<td>USUBJID</td>
<td>Record</td>
<td>Sequence</td>
<td>Variable(s)</td>
</tr>
<tr>
<td>0000022</td>
<td>At least one of the Seriousness criteria (AESCAN, AESCONG, AESDISAB, AESTHD, AESHOSP, AELIFE, AESOD or AESMIE) = &quot;Y&quot;, but AESER = &quot;N&quot; or empty.</td>
<td>fully executable</td>
<td>AE</td>
<td>01-701-1192</td>
<td>108</td>
<td>AESCAN, AESCONG, AESDISAB, AESTHD, AESER, AESHOSP, AELIFE, AESMIE, AESOD, AESDTLC, AETERM</td>
<td>N, N, N, N, N, Y, N,</td>
</tr>
<tr>
<td>0000022</td>
<td>At least one of the Seriousness criteria (AESCAN, AESCONG, AESDISAB, AESTHD, AESHOSP, AELIFE, AESOD or AESMIE) = &quot;Y&quot;, but AESER = &quot;N&quot; or empty.</td>
<td>fully executable</td>
<td>AE</td>
<td>01-701-1192</td>
<td>109</td>
<td>AESCAN, AESCONG, AESDISAB, AESTHD, AESER, AESHOSP, AELIFE, AESMIE, AESOD, AESDTLC, AETERM</td>
<td>N, N, N, N, N, Y, N,</td>
</tr>
<tr>
<td>0000022</td>
<td>At least one of the Seriousness criteria (AESCAN, AESCONG, AESDISAB, AESTHD, AESHOSP, AELIFE, AESOD or AESMIE) = &quot;Y&quot;, but AESER = &quot;N&quot; or empty.</td>
<td>fully executable</td>
<td>AE</td>
<td>01-701-1211</td>
<td>121</td>
<td>AESCAN, AESCONG, AESDISAB, AESTHD, AESER, AESHOSP, AELIFE, AESMIE, AESOD, AESDTLC, AETERM</td>
<td>N, N, N, N, N, Y, N,</td>
</tr>
<tr>
<td>0000022</td>
<td>At least one of the Seriousness criteria (AESCAN, AESCONG, AESDISAB, AESTHD, AESHOSP, AELIFE, AESOD or AESMIE) = &quot;Y&quot;, but AESER = &quot;N&quot; or empty.</td>
<td>fully executable</td>
<td>AE</td>
<td>01-704-1008</td>
<td>312</td>
<td>AESCAN, AESCONG, AESDISAB, AESTHD, AESER, AESHOSP, AELIFE, AESMIE, AESOD, AESDTLC, AETERM</td>
<td>N, N, N, N, N, N, N,</td>
</tr>
</tbody>
</table>
CORE – FORMEDIX

Formedix CORE is a free, downloadable Windows desktop application that provides an easy way to run validations on local data and identify standards conformance issues.
## xst validation report created on 2023-05-22 at 05:19:23

### Issue Summary

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Rule ID</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>TY</td>
<td>CORE-000260</td>
<td>VOTNUM is not among VOTNUM in TY domains.</td>
</tr>
<tr>
<td>MB</td>
<td>CORE-000264</td>
<td>Primary analysis used but MHDSOC5S and MHDSOC are not equal</td>
</tr>
<tr>
<td>AE</td>
<td>CORE-000002</td>
<td>AE_ID must be one of the followings (either AEID5A, AEID5B, AEID5C, AEID5D, AEID5A, AEID5B, AEID5C, AEID5D, AEID5A) + &quot;Y&quot;, but AE_ID = &quot;N&quot; or empty.</td>
</tr>
<tr>
<td>AE</td>
<td>CORE-000004</td>
<td>AEID5A is not empty and AEID5B is empty.</td>
</tr>
<tr>
<td>AE</td>
<td>CORE-000004</td>
<td>There is no one-to-one relationship between AEID5A and AEID5B.</td>
</tr>
<tr>
<td>AE</td>
<td>CORE-000004</td>
<td>AEID5C and AEID5D do not have a one-to-one relationship.</td>
</tr>
<tr>
<td>MB</td>
<td>CORE-000004</td>
<td>AID5A is not empty and AID5B is empty.</td>
</tr>
</tbody>
</table>

### Rules Reports

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Issue ID</th>
<th>Rule ID</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>TY</td>
<td>CORE-000260</td>
<td>VOTNUM is not among VOTNUM in TY domains.</td>
<td></td>
</tr>
<tr>
<td>MB</td>
<td>CORE-000264</td>
<td>Primary analysis used but MHDSOC5S and MHDSOC are not equal</td>
<td></td>
</tr>
<tr>
<td>AE</td>
<td>CORE-000002</td>
<td>AE_ID must be one of the followings (either AEID5A, AEID5B, AEID5C, AEID5D, AEID5A, AEID5B, AEID5C, AEID5D, AEID5A) + &quot;Y&quot;, but AE_ID = &quot;N&quot; or empty.</td>
<td></td>
</tr>
<tr>
<td>AE</td>
<td>CORE-000004</td>
<td>AEID5A is not empty and AEID5B is empty.</td>
<td></td>
</tr>
<tr>
<td>AE</td>
<td>CORE-000004</td>
<td>There is no one-to-one relationship between AEID5A and AEID5B.</td>
<td></td>
</tr>
<tr>
<td>AE</td>
<td>CORE-000004</td>
<td>AEID5C and AEID5D do not have a one-to-one relationship.</td>
<td></td>
</tr>
<tr>
<td>MB</td>
<td>CORE-000004</td>
<td>AID5A is not empty and AID5B is empty.</td>
<td></td>
</tr>
</tbody>
</table>
CORE – FORMEDIX RECAP

Features
> Load data from local XPT datasets
> Select subset of datasets to validate
> Select rule set
> Select subset of rules to run
> Refresh rule sets from CDISC Library
> Sorting/filtering of results
> View Excel results report

Benefits
> Early visibility of SDTM conformance
> Early visibility of safety and efficacy data
> Make informed decisions early on
> Run more trials, focus on those showing promise
> Ultimately, speed the delivery of life changing treatments

Future functionality
> Final production version of CORE engine
> Final production version of initial rulesets (e.g. SDTM-IG 3.4)
> Additional rulesets (SEND, ADaM, FDA business rules etc)
> Load data from CSV, Dataset-XML, Dataset-JSON
> Validate against MedDRA / WHODrug and other dictionaries
> (maybe) Linux / MacOS support
CORE – PINNACLE 21

P21 supports CDISC Open Rules Engine (CORE), enabling execution of machine-readable CDISC Conformance Rules via desktop GUI or CLI on any P21 Community platform.
## Problems with your Validator installation detected which may cause inaccurate validation results

### Experimental Engine Used

This experimental engine is for developmental experimentation only. Use at your own risk. It has a limited rule set and thus cannot detect the majority of data issues (false negatives). No regulatory agency uses this engine. For actual submissions, use a valid production engine (FDA, PMEA, etc.). Refer to link for details.

### Table: Pinnacle 21 Validator Report

<table>
<thead>
<tr>
<th>Source</th>
<th>Rule ID</th>
<th>Message</th>
<th>Severity</th>
<th>Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORE</td>
<td>CS530000041</td>
<td>At least one of the Seriousness criteria (AESCON, AESCON, AESCON) is Y.</td>
<td>Error</td>
<td>126</td>
</tr>
<tr>
<td>CUSTOM</td>
<td>CS530000042</td>
<td>AESCON is empty.</td>
<td>Error</td>
<td>473</td>
</tr>
<tr>
<td>DOM</td>
<td>CS530000043</td>
<td>DOM is not empty.</td>
<td>Error</td>
<td>1191</td>
</tr>
<tr>
<td>CM</td>
<td>CS530000044</td>
<td>Missing value for CMDS02. when CMDS02, CMDS02XT or CMDS02ST is provided</td>
<td>Warning</td>
<td>82</td>
</tr>
<tr>
<td>CUSTOM</td>
<td>CS530000045</td>
<td>DOM is not empty.</td>
<td>Error</td>
<td>751</td>
</tr>
</tbody>
</table>

### Table: Pinnacle 21 Validator Report

<table>
<thead>
<tr>
<th>Source</th>
<th>Rule ID</th>
<th>Message</th>
<th>Severity</th>
<th>Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORE</td>
<td>CS530000041</td>
<td>At least one of the Seriousness criteria (AESCON, AESCON, AESCON) is Y.</td>
<td>Error</td>
<td>126</td>
</tr>
<tr>
<td>CUSTOM</td>
<td>CS530000042</td>
<td>AESCON is empty.</td>
<td>Error</td>
<td>473</td>
</tr>
<tr>
<td>DOM</td>
<td>CS530000043</td>
<td>DOM is not empty.</td>
<td>Error</td>
<td>1191</td>
</tr>
<tr>
<td>CM</td>
<td>CS530000044</td>
<td>Missing value for CMDS02. when CMDS02, CMDS02XT or CMDS02ST is provided</td>
<td>Warning</td>
<td>82</td>
</tr>
<tr>
<td>CUSTOM</td>
<td>CS530000045</td>
<td>DOM is not empty.</td>
<td>Error</td>
<td>751</td>
</tr>
</tbody>
</table>

---

**CORE – PINNACLE 21 Report**
CORE – PINNACLE 21 RECAP

Experimental, In-development Engine

> CORE has a limited rule set and thus cannot detect most data issues. Expect "false negatives," meaning, your datasets will deceptively appear to have fewer issues than they actually do.

> More important, because no regulatory agency uses the CORE Engine, you should instead always use a valid production-grade Engine (FDA, PMDA, etc.) for actual submissions of real study data.

How to Run CORE Engine in P21 Community

> Via the desktop GUI, go to Validator, set Engine to CDISC (0000.1) and choose a Configuration, e.g., SDTM-IG 3.4 (CDISC), which is the only one available from CDISC at this time. Drop your files into the Source Data box, and then Validate.

> Via the CLI.
YAML is a human-readable data serialization language that is often used for writing configuration files.

YAML is a popular programming language because it is designed to be easy to read and understand. It can also be used in conjunction with other programming languages. Because of its flexibility and accessibility.
YAML – SYNTAX

> YAML files use a .yml or .yaml extension and follow a specific syntax rules.

> YAML has features that come from Perl, C, XML, HTML, and other programming languages. YAML is also a superset of JSON, so JSON files are valid in YAML.

> There are no usual format symbols, such as braces, square brackets, closing tags, or quotation marks.

#Comment: This is a supermarket list using YAML
#Note that - character represents the list
---
food:
- vegetables: tomatoes  #first list item
- fruits:  #second list item
citrics: oranges
tropical: bananas
nuts: peanuts
sweets: raisins
Conformance Rule Specification Development

Human-readable Specification

<table>
<thead>
<tr>
<th>Rule ID</th>
<th>SDTM/IG</th>
<th>Rule Version</th>
<th>Class</th>
<th>Domain</th>
<th>Variable</th>
<th>Condition</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG0025</td>
<td>3.4</td>
<td>1</td>
<td>ALL</td>
<td>ALL</td>
<td>VISITDY</td>
<td>VISITNUM IS NOT in TV.VISITNUM</td>
<td>VISITDY = null</td>
</tr>
</tbody>
</table>

Document Section Item Cited Guidance

ig v3.4 4.4.5 VISITDY must not be populated for unplanned visits, since VISITDY is, by definition, the planned study day of visit, and since the actual study day of an unplanned visit belongs in a –DY variable.

Authoring Sources:
- CDISC Standards
- FDA Validation Rules
- Community proposals

Conformance Rule Specification Development

Core Rule Development

Executable Rule (YAML) in CORE Rule Editor

Rule developed and tested in CORE Rule Editor and CORE Engine

Centralized

Specifications Catalog

CDISC Library

Publish
RULES – EXAMPLE: SDTM.CG0026.yml

Authority:
Organization: CDISC

Check:
   all:
      - name: "--TPTREF"
        operator: empty
      - name: "--RFTDTC"
        operator: exists
      - name: "--RFTDTC"
        operator: non_empty

Citations:
- Cited Guidance: "The actual date or date and time of a time point that acts as
  a fixed reference for a series of planned time points, represented in a standardized
  character format. The fixed reference point is in --TPTREF"

   Document: Model v2.0
   Item: "--RFTDTC"
   Section: Timing

Core:
   Id: CDISC.SDTMIG.CG00026
   Version: '1'
   Description: "Trigger error when --TPTREF = null and --RFTDTC is populated"

Outcome:
   Message: "--RFTDTC is populated when --TPTREF is null"

References:
- Origin: SDTM and SDTMIG Conformance Rules
  Rule Identifier:
     Id: CG0026
     Version: '1'
     Version: '2.0'
   Rule Type: Range & Limit

Scopes:
   Classes:
      Include:
         - All
   Domains:
      Include:
         - All
   Standards:
      - Name: SDTMIG
        Version: '3.4'
   Sensitivity: Record
   Severity: Error
RULES – EXAMPLE: CUSTOM RULE

Pinnacle 21 Validator Report

<table>
<thead>
<tr>
<th>Source</th>
<th>Rule ID</th>
<th>Message</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE</td>
<td>CDISC:SDTM.CG00041</td>
<td>At least one of the Seriousness criteria (AESCAN, AESCONG, AESSISAB, AESIDTH, AESHOSP, AESLIFE, AESOD or AESMIE) = &quot;Y&quot;, but AESER = &quot;N&quot; or empty.</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>CUSTOM:SDTM.CG00039</td>
<td>AESNDTC is empty</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>CUSTOM:SDTM.CG00037</td>
<td>DOMAIN is not empty</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>CDISC:SDTM.CG00084</td>
<td>AESDOSYS is not empty and AESDOSYCD is empty</td>
<td>Warning</td>
</tr>
<tr>
<td></td>
<td>CDISC:SDTM.CG00487</td>
<td>AESDTDC is present in a Findings general observation class</td>
<td>Warning</td>
</tr>
<tr>
<td>CM</td>
<td>CDISC:SDTM.CG0032</td>
<td>Visit Day cannot be found in TrialVisit (TV) domain</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>CDISC:SDTM.CG0014</td>
<td>Missing value for CMDOBU, when CMDOUSE, CMDOOSTXT or CMDOSTOT is provided</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>CUSTOM:SDTM.CG0001</td>
<td>DOMAIN is not empty</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>CDISC:SDTM.CG0007</td>
<td>CMSTDTC is present in a Findings general observation class</td>
<td>Warning</td>
</tr>
<tr>
<td>DM</td>
<td>CDISC:SDTM.CG0029</td>
<td>RFENDTC is missing when ARM is provided.</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>CUSTOM:SDTM.CG00032</td>
<td>DOMAIN is not empty</td>
<td>Error</td>
</tr>
</tbody>
</table>

CUSTOM:SDTM.CG0032 Domain is not empty Verify that DOMAIN is not empty Presence Error
CONCLUSION
Why CORE?

- OPEN SOURCE
- CUSTOM EXTENSION
- CUSTOM RULES
- MULTIPLE SOFTWARE
How to Volunteer

https://www.cdisc.org/volunteer/form

> Select CORE Rules Team

> Expected Engagement

  o Time Period: 3 - 6 months, or longer, if able
  o Hours per week: 2 - 4 hours, in addition to meeting attendance
  o Weekly Meetings and Workshops: Recorded and available for review
  o Rules Development Workshop Tuesdays 9am - 11am ET
  o Weekly Team Meeting – Thursdays 11am - 12pm ET
Homemade CORE
Why?

> YAML Rule Handling: I wanted to understand how Python handles YAML rules and parsing.
> CMD Environment: I aimed to create a command-line environment that allows users to conveniently check and validate their data.
> Practical Application: The example serves as a demonstration of how Python can be used to apply YAML rules to a given dataset.
> Information Sharing: Through this example, I aimed to provide insights and knowledge to those interested in YAML parsing in Python.
PACKAGES

- Pandas
- Yaml
- Template
- Argparse

```python
1 import pandas as pd
2 import yaml
3 from jinja2 import Template
4 import argparse
```
PARSE COMMAND-LINE ARGUMENTS

```python
# Parse command-line arguments
parser = argparse.ArgumentParser()
parser.add_argument('csv_path', help='Path to the CSV dataset')
parser.add_argument('--output', help='Path to the output CSV file', default='output.csv')
args = parser.parse_args()
```

- **parser = argparse.ArgumentParser():**
  It creates an instance of the ArgumentParser class from the argparse module. The ArgumentParser class provides a way to specify the arguments that the program expects.

- **parser.add_argument('name_parameter', help='Description_of_the_parameter'):**
  This line adds a positional argument named `name_parameter` to the argument parser. The help parameter is used to provide a description of the argument for the user.

- **args = parser.parse_args():**
  This line parses the command-line arguments provided by the user and stores the values in the args variable.
# Parse command-line arguments
parser = argparse.ArgumentParser()
parser.add_argument('csv_path', help='Path csv dataset')
parser.add_argument('--output', help='Path to the output CSV file', default='output.csv')
args = parser.parse_args()

# Import csv dataset
df = pd.read_csv(args.csv_path)
**DEF: CHECK_CONDITIONS**

- **Check_conditions** takes two parameters:
  - Conditions;
  - Data.
- FOR loop that iterates over each condition in the conditions list.
- Inside the loop, the code extracts the values:
  - Field;
  - Operator;
  - Value from each condition.
- If the operator is 'exists', it checks if the field exists in the data dictionary. If the field is not present in the data dictionary, the function returns False to indicate that the conditions are not met.
- If the operator is 'less_than_or_equal', it uses a templating approach to evaluate the value. It renders the value by substituting variables from the data dictionary using a template engine. It then compares the rendered value with the corresponding value in the data dictionary for the given field. If the value in the data dictionary is greater than the rendered value, the function returns False.
- If none of the conditions evaluated so far have returned False, the function reaches the end of the loop and returns True, indicating that all conditions have been met.

```python
# Check if all conditions of the rule are satisfied

def check_conditions(conditions, data):
    for condition in conditions:
        field = condition['field']
        operator = condition['operator']
        value = condition.get('value')  # Value is optional for some operators

        if operator == 'exists':
            if field not in data:
                return False

        elif operator == 'less_than_or_equal':
            template = Template(value)
            rendered_value = template.render(data)
            if data[field] > rendered_value:
                return False

    return True
```
RULE01.YAML

> Name of the Rule: **StartDateCannotBeGreaterThanEndDate**

> Conditions: specify the requirements that need to be met for the rule to be considered valid.

  o It checks the existence of the "start_date" field.
  o It checks the existence of the "end_date" field.
  o It compares the value of the "start_date" field with the "end_date" field using the "less_than_or_equal" operator.

> Operator: **less_than_or_equal**: This line indicates the comparison operator used for the condition.

> value: "{{ end_date }}": It specifies the value to compare against. It uses a template expression "{{ end_date }}" that will be rendered dynamically during runtime.

> Message: Start date cannot be greater than end date.
It defines the error message associated with this rule. If the conditions specified earlier are not met, this error message will be triggered.
OPEN YAML RULE

This code reads the contents of a YAML file named "rule.yaml" and loads it into a Python data structure using the yaml.safe_load() function from the PyYAML library.

> With open('rule.yaml') as file: This line opens the file named "rule.yaml" in the current directory using the open() function. It assigns the file object to the variable file.

> Rules = yaml.safe_load(file): This line uses the yaml.safe_load() function from the PyYAML library to parse the contents of the file. It takes the opened file object file as the argument and returns a Python data structure representing the YAML data. The parsed data is assigned to the variable rules.

> Rules: It contains the Python data structure representing the YAML data from the "rule.yaml" file.
**APPLY RULES**

*This process a dataset represented by a DataFrame “df” and applies a set of rules to filter out observations that do not meet the conditions specified by the rules.*

- The code iterates over each observation in df using a for loop.
- **Satisfies_rule**: It tracks whether the current observation satisfies any of the rules.
- Nested loop: The second loop iterates over each rule in the rules list.
- **Check_conditions**: it evaluates whether the current observation satisfies the conditions specified by the current rule.
- If conditions satisfied for the current rule then **satisfies_rule** = True and breaks out of the nested loop.
- After the nested loop, the code checks whether satisfies_rule is still False. If it is, it means that the current observation did not satisfy any of the rules. In this case, the observation is appended to the new_dataset list using new_dataset.append(observation).
- **New_dataset** contain only those observations that did not satisfy any of the rules.
> Convert list (df) to dataframe.
> Export as .csv

```python
# Print the new dataset
df1 = pd.DataFrame(df)
df1.to_csv(args.output, index=False)
```

**Example**

```
> outputtest.csv ×
1 start_date,end_date,other_field,num
2 2023-06-15,2023-06-13,value,1
3 2023-06-21,2023-06-20,value,2
```
**EXAMPLE**

**Terminal**

```bash
(venv) davidemarinucci@MBA-Project % python main.py dataset.csv --output outputtest.csv
```

**Output**

```
<table>
<thead>
<tr>
<th>start_date, end_date, other_field, num</th>
</tr>
</thead>
<tbody>
<tr>
<td>2023-06-15, 2023-06-13, value, 1</td>
</tr>
<tr>
<td>2023-06-21, 2023-06-20, value, 2</td>
</tr>
</tbody>
</table>
```
CONCLUSION

**PROs**

- We are able to produce a data check report in every stage of our study;
- It gives us the opportunity to create our own custom rules;
- It doesn’t require an .xpt version of the dataset as per Pinnacle 21 in order to produce data checks;
- Accessible through cmd;
- Open source.

**CONs**

- It requires YAML and PYTHON knowledge;
- Deep dive into the definition of the rules;