A tool for validation and vetting of model results in IAM COMPACT

Jan Ivar Korsbakken (jan.ivar.korsbakken@cicero.oslo.no)
CICERO Center for International Climate Research

What is `iam-validation`?

A package to help fix names and values in IAMC -formatted model output

- Checks variable, region, model, scenario names against definition lists (*like* `nomenclature`*, with some extra feedback)*
- Checks values against defined ranges or reference data, with detailed feedback on deviations (like `pathways-ensemble-analysis`, with major extensions)

A tool to help harmonize names and assumptions in multi -model projects

- Gives detailed overview of all unrecognized model, scenario, region, variable names, and variable/unit combinations
- Flags whether model results pass given vetting criteria or adhere to reference data
- Helps produce detailed reports of where and by how much models deviate from reference timeseries data or single -valued vetting criteria

What are the main features?

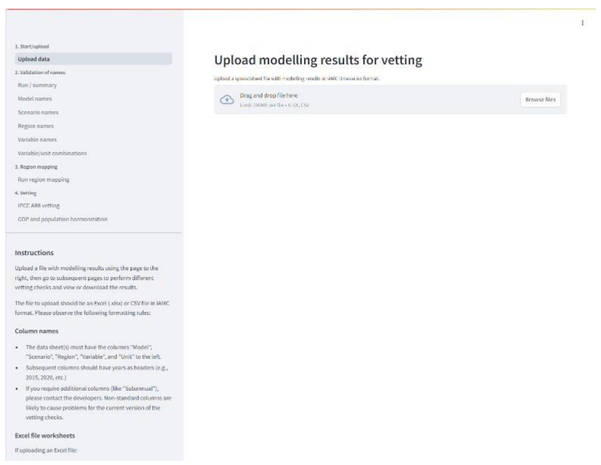
- Simple loading of data structure definitions (DSDs) and region mappings (*convenient wrapper for* `nomenclature` *repository loading)*
- Merge DSDs and override duplicate definitions in prioritized order
  - Useful for defining project-specific overrides and additions to the definitions in the `common-definitions` repo
- Get lists or DataFrames of all unrecognized model, scenario and variable names, and variable/unit and model/region combinations
- Define reference time series for multiple variables and regions, conveniently calculate detailed deviations per model, scenario, and year (major extension of the `Criterion` class in `pathways-ensemble-analysis`)
- CriterionTarget class to define targets and allowed ranges
- Output full deviation data as styled DataFrames, or export to Excel, with coloured highlighting of values outside target range

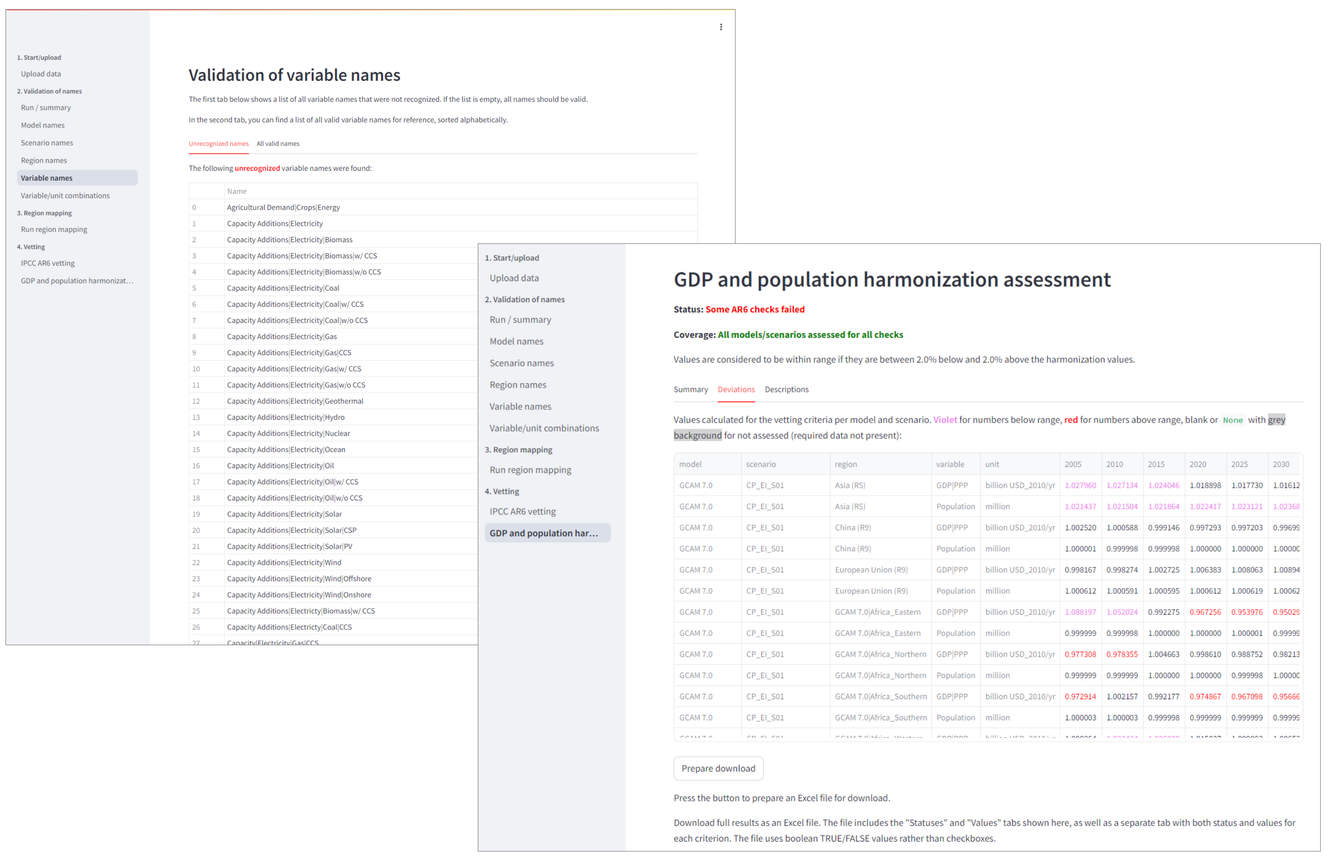
Why and how has it been used ?

Spinoff from Horizon Europe project *IAM* COMPACT (https://www.iam-compact.eu, grant #101056306)

Multi-model project, needed tool for cross-model validation.

Developed a Python library, and GUI web app. Python library refactored into a project-independent package, `iam-validation` (GUI still project-specific).





How to: Find invalid variable/region/model/scenario names

```python
from iam_validation.nomenclature import NomeclatureDefs, MergedDefs, COMMON_DEFINITIONS_URL
from pathlib import Path
import pyam, pandas as pd

# Get variable and region names from `common_definitions`
common_defs = NomeclatureDefs.from_url(COMMON_DEFINITIONS_URL, region_mappings=False)

# Get project-specific names for all dimensions, and region-mappings
project_defs = NomenclatureDefs.from_url(repo_url, dimensions=['model', 'scenario',
                                                               'region', 'variable'])
# Merge the definitions, and use project-specific definitions where they overlap
merged_defs: MergedDefs = common_defs.update(project_defs)

# Load a file with model results as an IAMC-formatted Excel or CSV file.
model_df = pyam.IamDataFrame('./my_model_output.xlsx')

# Get invalid names, including recognized model-native ones. Returns a dict with dimension
# names as keys, and invalid names or combos as lists or DataFrames
invalid_names: dict[str, list[str]|pd.DataFrame] = \
    merged_defs.get_invalid_names(model_df, raw_model_regions=True)

# Get invalid unit/variable combos. Returns a DataFrame, with one row for each known
# variables that has unrecognized units, and columns with lists of the unrecognized units for
# each variable, and one lists of the valid unit names for that variable.
invalid_units: pd.DataFrame = merged_defs.get_invalid_variable_units(model_df)
```

How to: Write formatted output to Excel, with highlighted deviations

```python
from iam_validation.output import TimeseriesRefTargetOutput
import pandas as pd
from pandas.io.formats.style import Styler

# Create an outputter object that can produce styled output, using the `CriterionTargetRange`
# object `target_range` and the model data `model_df` from the previous example.
# `TimeseriesRefTargetOutput.prepare_styled_output` gives us a dict with two elements:
# - "summary": A styled pandas DataFrame with the maximum deviation per variable,
#   model, scenario and region.
# - "full_comparison": A wide styled DataFrame with the ratios for each variable, model,
#   scenario, region, and *year*.
# The DataFrames have colored highlights for the ratios that fall outside the allowed range.
ratios_outputter = TimeseriesRefTargetOutput(target_range)
ratios_styled_dfs: dict[str, Styler] = ratios_outputter.prepare_styled_output(model_df)

# Write the output to an Excel file, which will have colored cells for deviations outside the
# range. The file will have two worksheets worksheets, one with summary and one with full
# comparison (mirroring the dict produced in the previous step).
ratios_outputter.to_excel('comparison.xlsx', results=ratios_styled_dfs)
```

How to: Compare with time series reference data

```python
from iam_validation.criteria import TimeseriesRefCriterion
from iam_validation.targets import CriterionTargetRange
import pyam, pandas as pd

# Load your reference timeseries data for one or more variables
ref_data = pyam.IamDataFrame('./my_reference_data.xlsx')

# Create a reference object that computes the ratio of model results to reference
# data and define allowed target range of +/- 5%.
timeseries_ref = TimeseriesRefCriterion(criterion_name='Reference data ratio comparison',
    reference=ref_data, comparison_function='ratio', broadcast_dims=('model', 'scenario'))
target_range = CriterionTargetRange(criterion=timeseries_ref, target=1.0, range=(0.95, 1.05))

# Load the model data that you want to compare to the reference data
model_df = pyam.IamDataFrame('./my_model_output.xlsx')

# Get a pandas.Series with the ratios of model data relative to reference data, for each data
# point that exists in both the model output and reference data
comparison_ratios: pd.Series = timeseries_ref.get_values(model_df)

# Get a pandas.Series with True/False for what data points are inside the allowed target
# range.
in_range_statuses: pd.Series = target_range.get_in_range(model_df)
```

Where is the code (and in what state) ?

`iam-validation` GitHub repo: https://github.com/ciceroOslo/iam-validation

- Work in progress
- Docs (as they are) in the `/docs/build` folder, will eventually be on ReadTheDocs. Nothing yet except autogenerated from docstrings!
- Test coverage only for a limited part of the code
- Guides and better documentation of class structure to be added
- Feel free to open issues if you have questions or feature suggestions!

IAM COMPACT validation app
Code: https://github.com/ciceroOslo/iamcompact-validation-ui
Deployed app: https://validation.iam-compact.eu/

- NB! Specific to IAM COMPACT, will not currently work with other projects
- …but feel free to check the code or the app for inspiration
- …or get in touch if you want to make something like it for your project

Acknowledgements

- Initial development funded through the EU Horizon Europe project IAM COMPACT (grant agreement no. 101056306)
- `iam-validation` subclasses or derives some code from `nomenclature-iamc` and `pathways-ensemble-analysis`
- IAM COMPACT validation web app builds in part on code by George Xexakis at HOLISTIC, developed under the EU Horizon Europe project DIAMOND