emodpy-generic

Institute for Disease Modeling

Mar 01, 2022

CONTENTS

The HIV Workflow (hiv_workflow) is a set of scripts and libraries utilizing idmtools, idmtools-calibra, and emodpy-hiv for calibration and running of EMOD.

CHAPTER

ONE

DOCUMENTATION

Documentation is available at https://docs.idmod.org/projects/hiv_workflow/en/latest/index.html.

CHAPTER

TWO

INSTALLATION

2.1 Software prerequisites

- This guide assumes Python 3.7.X (3.7.7 or higher) (64-bit) is installed (https://www.python.org/downloads/release/python-377) in Windows or Linux and assumes it is custom-installed into C:Python37 in Windows or /c/Python37 in Linux. (Download the exe or tgz to install.)
- This guide further assumes a Linux-like command terminal is being used for Windows (e.g. git bash in Windows), not the built-in Windows cmd.

2.2 Installation steps

1. Create and activate a clean virtual environment.

   If you have Python installed to a different directory, please update the Python interpreter path below to match your installation of Python.

   1. Create the virtual environment:

      ```
      /c/Python37/python -m venv ~/environments/hiv_workflow
      ```

   2. Activate the virtual environment:

      - Bash in Windows:

        ```
        source ~/environments/hiv_workflow/Scripts/activate
        ```

      - In Linux:

        ```
        source ~/environments/hiv_workflow/bin/activate
        ```

   3. Ensure pip is up-to-date (the default version has major issues):

      ```
      python -m pip install pip --upgrade
      ```

2. Obtain and install hiv_workflow:

   ```
   git clone https://github.com/InstituteforDiseaseModeling/hiv_workflow.git
   cd hiv_workflow
   python -m pip install --upgrade pip
   pip install . -r requirements.txt
   ```

3. Verify the installation:

```
cd examples/rakai-optimtool
python -m hiv_workflow.scripts.available_parameters -m baseline
```

CHAPTER

THREE

SAMPLE WORKFLOW

The examples/rakai-optimtool/demo.md file contains example commands that highlight a simple but typical workflow linking the scripts contained in hiv_workflow. These commands are expected to be run with your working directory being examples/rakai-optimtool (the root of the example project).

CHAPTER

FOUR

KNOWN ISSUES

- If running pip install results on this error: 'ERROR: THESE PACKAGES DO NOT MATCH THE HASHES FROM THE REQUIREMENTS FILE'
  - Try running adding `--no-cache-dir` as part of your pip install command. i.e. `pip install -e . -r requirements.txt --no-cache-dir`
  - Issue 26
- If running pip install results on dependencies issues:
  - Try updating pip: `python -m pip install --upgrade pip`
- If when running one of the demo samples you get a "hiv_workflow not found" or even "Python not found":
  - Try activating your virtual environment as shown above

4.1 Project workflow

Contents

4.1.1 Scripts

Run scripts in the following format:

```
python -m hiv_workflow.scripts.<script_name>(no .py) <script arguments>
```

calibrate.py

Calibration tool for EMOD-HIV. For an example, see calibration in the Rakai optimtool demo.

resample.py

Generates parameter sets/samples from a directory of calibration results generated by calibrate.py. For an example, see sampling in the Rakai optimtool demo.

run.py

General simulation run tool. Can be used:

- With one or more models (scenarios) at a time.
- With or without samples generated by resample.py (uses builder defaults without samples).
- With or without a sweep Python file (uses builder/sample defaults if not sweeping).

Generates one experiment per sweep parameter set per model. Each experiment contains one simulation per sample (one, the default, if no samples). For an example, see running in the Rakai optimtool demo.

download.py

Downloader of suites of simulations that is:

- Called by -d flag of run.py (if specified), or on its own.
- Called with -id (suite_id) or -r <receipt_path> for more descriptive output format.

For an example, see download on demand in the Rakai optimtool demo.

available_parameters.py

Discovers and displays all parameters available for calibration/scenario modification in a specified model. For an example, see parameter verification in the Rakai optimtool demo.

4.1.2 Components that have/are APIs (definition requirements)

- Project manifest.py
- Sweep files
- Models
  - Subdirectory __init__.py files
  - Builders (campaign, config, demographics)
    - Builder function signature
    - Parameter setting function signatures
    - Parameter setting function signature defaults
- Site data (analyzers, reference data, parameters to calibrate, site scaling info)

4.1.3 Project directory structure

A project directory must have these components:

- Items in <> are required and have names that can vary by project
- Items in () are optional, depending on context
- Items in (<>) are optional, and their names can vary project
- Items in {} are simply suggested structure and names

```
<project_dir>
|
|- manifest.py
|
|- models
|   |
|   |- __init__.py
|   |
|   |- <model_dir_1>
|   |   |
|   |   |- __init__.py
|   |   |- (<campaign.py>)
|   |   |- (<config.py>)
|   |   |- (<custom_sample_handling.py>)
|   |   |- (<demographics.py>)
|   |  ...
|   |- <model_dir_N>
|   |   |
|   |   |- __init__.py
|   |   |- (<campaign.py>)
|   |   |- (<config.py>)
|   |   |- (<custom_sample_handling.py>)
```

(continues on next page)

(continued from previous page)

```
|   |   |- (<demographics.py>)
|
|- {bin}
|
|- {calibration}
|   |
|   |- {ingest_forms}
|   |   |
|   |   |- <calibration_ingest_form.xlsm>
|   |
|   |- {output}
|   |   |
|   |   |- {calibration_directory_1}
|   | ...
|   |   |- {calibration_directory_N}
|   |
|   |- {samples}
|   |   |
|   |   |- <calibration_samples_1.csv>
|   |   | ...
|   |   |- <calibration_samples_N.csv>
|
|- {data}
|   |
|   |- {raw}
|   |   |
|   |   | - (<raw_data_file_1>)
|   |   | ...
|   |   | - (<raw_data_file_N>)
|   |
|   |- {processed}
|   |   |
|   |   | - (<processed_data_file_1>)
|   |   | ...
|   |   | - (<processed_data_file_N>)
|
|- (<model_input_files>)    # for back-compatibility, a place for EMOD demographics␣
↪files
|
|- {results}
|   |
|   |- {sweeps.py}
|   |
|   |- {scenario_results_1}
|   |   |
|   |   |- {output}
```

4.1.4 manifest.py

Minimum attribute requirements:

asset_collection_of_container Path of file with asset collection id of container to run in (if run environment uses containers).

executable_path Path to local model executable, or if using containers, the name of the executable within the container.

schema_path Path to the model schema file, or path to where it will be at runtime.

post_processing_path "Standard" to use the built-in EMOD Python post-processor, path to a custom post-processor, or "None" for no post-processing (currently calibration-only).

in_processing_path "Standard" to use the built-in EMOD Python in-processor, path to a custom in-processor, or "None" for no in-processing. (There is currently no "Standard" in-processor).

pre_processing_path "Standard" to use the built-in EMOD Python pre-processor, path to a custom pre-processor, or "None" for no pre-processing. (There is currently no "Standard" pre-processor).

ingest_filename Path to xlsm file of parameter/analyzer/site/reference data for calibration. calibrate.py could be refactored to take alternate input.

eemographics_paths List of paths of existing EMOD demographics files. Only if using demographics file(s) for back-compatibility.

4.1.5 models/__init__.py

Requirements: None

4.1.6 models/<model_dir_N>/__init__.py

Minimum attribute requirements:

config_builder

- A reference to the Python function that builds the model default config(.json) object via emodpy.
- Minimum signature requirements for config setting function (note function name is not required): `def config_builder(schema_path, max_memory_mb=None, demographics_paths=None, params_to_modify=None, dry_run=False, **kwargs):`
- Returns: config (object), available_parameters (list)

demographics_builder

- A reference to the Python function that builds the model default demographics(.json) object via emodpy.
- Minimum signature requirements for demographics builder function (note function name is not required): `def build_demographics(manifest, params_to_modify=None, dry_run=False, **kwargs):`
- Returns: demographics (object), available_parameters (list)

campaign_builder

- A reference to the python function that builds the model default campaign(.json) object via emodpy.
- Minimum signature requirements for campaign builder function (note function name is not required): `def build_campaign(schema_path, params_to_modify=None, dry_run=False, **kwargs):`
- Returns: campaign (object), available_parameters (list)

custom_sample_constrainer

- A reference to the Python function that enforces logical constraints on parameters to avoid nonsensical parameterizations.
- Exact signature requirement for sample constraining function (note function name is not required): `def constrain_sample(sample):`

kwargs

- A dictionary of key: value elements that are passed to all builders (above) of the model. This dict may be empty, but is useful for synchronization of actions between the builders (e.g., if adding a certain campaign item requires a config update, one could trigger both actions by using a kwargs key/value conditional in the config and campaign builder logic).

4.1.7 Calibration sample csv files

- For use with run.py.
- These are output files generated by resample.py.

4.1.8 Sweep Python files

For use with run.py

Minimum attribute requirements:

parameter_sets:

- A dict of model_name entries. Each value is a dict:

```
{
    'sweeps': iterable (list, generator, etc), # REQUIRED
    'experiment_name': <some experiment name>  # OPTIONAL
}
```

Iterables should contain or evaluate to dicts of parameter: value entries, used as parameter overrides (sweeps) by run.py. The result will be one simulation per override set per base sample.

If specified, the given experiment name will be used for each sweeping parameter set (one experiment each). The default experiment name is the model_name.

For example:

```
parameter_sets = {
    'baseline': {
        'experiment_name': 'Some alternate baseline name',
        'sweeps': [
            {'csw_female_uptake_coverage': 0.2, 'csw_male_uptake_coverage': 0.1},
            {'risk_assortivity_informal': 0.9},
            {'csw_female_uptake_coverage': 0.3, 'csw_male_uptake_coverage': 0.1},
            {'csw_female_uptake_coverage': 0.3, 'csw_male_uptake_coverage': 0.2},
        ]
    },
    'scenario1': {
        'sweeps': [
            {'art_duration': 20000},
            {'art_duration': 30000}
        ]
    }
}
```

The result of this definition means that each model (scenario) can be run with either the same or different sweeping parameter sets. Individual entries in "sweeps" are independent from each other. There is no requirement that they have any overlap in parameters or values specified, so long as parameter names and values are valid.

4.1.9 Raw and processed data files

These depend entirely on the project, but are intended for loading during e.g. demographics configuration setting.

4.2 API reference

4.2.1 hiv_workflow package

Subpackages

hiv_workflow.lib package

Subpackages

hiv_workflow.lib.analysis package

Submodules

hiv_workflow.lib.analysis.age_bin module

class hiv_workflow.lib.analysis.age_bin.**AgeBin**(*start*, *end*, *delimiter=None*)

Bases: object

exception InvalidAgeBinFormat

Bases: Exception

exception NotMergeable

Bases: Exception

STR_FORMAT = '[%s%s%s)'

SPLIT_REGEX = re.compile('^\\[(?P<start>[0-9.]+)(?P<delimiter>[^0-9.]+)(?P<end>[0-9.]+

DEFAULT_DELIMITER = ':'

ALL = 'all'

merge(*other_bin*)
Create a single AgeBin representing two adjacent AgeBins. Keeps delimiter of 'self'. :param other_bin: merge self with this other AgeBin object (self is lower age than other_bin) :return: an AgeBin object with delimiter set to self.delimiter (not other_bin.delimiter)

contains(*other_bin*)
Is other_bin contained within the bounds of self? :param other_bin: an AgeBin object :return: True/False

to_tuple()

classmethod from_string(*str*)

classmethod merge_bins(*bins*)

classmethod can_upsample_bins(*bins*, *target_bin*)

hiv_workflow.lib.analysis.base_distribution module

class hiv_workflow.lib.analysis.base_distribution.**BaseDistribution**
Bases: object

exception UnknownDistributionException
Bases: Exception

LOG_FLOAT_TINY = -708.3964185322641

abstract prepare(*dfw:* hiv_workflow.lib.analysis.data_frame_wrapper.DataFrameWrapper, *channel: str*, *weight_channel: str*, *additional_keep: List[str]*) → *hiv_workflow.lib.analysis.data_frame_wrapper.DataFrameWrapper*
Prepare a DataFrameWrapper and this distribution object for a compare() call together. This includes dataframe verification/data checking, adding additional distribution-specific channels/columns, and trimming the data columns to the minimum needed. Depending on the particular distribution type, additional attributes on self may be set to prepare it in addition to the dfw (e.g. setting self.alpha_channel and self.beta_channel, derived from arg channel on self for BetaDistribution). :param dfw: DataFrameWrapper containing data that will be used in a future compare() call :param channel: data channel/column in dfw that the future compare() call be regarding :param weight_channel: an analyzer weighting channel that must be kept, if specified :param additional_keep: additional columns in the DataFrameWrapper to preserve

Returns: a modified copy of the input DataFrameWrapper

abstract compare(*df: pandas.core.frame.DataFrame*, *reference_channel: str*, *data_channel: str*) → float
Returns a score between -708.3964 and 100 (bad, good) for how well the dataframe (df) simulation data column (data_channel) matches the reference data column (reference_channel). :param df: pandas DataFrame with columns of data to compare :param reference_channel: reference data channel in dataframe :param data_channel: simulation data channel to compare to the reference data channel

Returns: a floating point score measuring the degree of data/reference fit, also known colloquially as 'likelihood'

abstract add_percentile_values(*dfw:* hiv_workflow.lib.analysis.data_frame_wrapper.DataFrameWrapper, *channel: str*, *p: float*) → List[str]
Adds a new data channel to a DataFrameWrapper object that represents a requested probability threshold/value for a specified channel. Useful for creating uncertainty envelopes in plots. :param dfw:

DataFrameWrapper with data to construct percentiles and to add percentiles to :param channel: the column in dfw that percentiles will be constructed from/for :param p: the 0-1 percentile level for the given channel to add

Returns: a list containing the new channel name in dfw

classmethod from_string(*distribution_name: str*) → *hiv_workflow.lib.analysis.base_distribution.BaseDistribution*
Loads and returns a distribution object of the type appropriate to the provided name, e.g. BetaDistribution from "beta". :param distribution_name: name of distribution type to load

Returns: a distribution object

classmethod from_uncertainty_channel(*uncertainty_channel: str*) → *hiv_workflow.lib.analysis.base_distribution.BaseDistribution*
Loads and returns a distribution object of the type appropriate to the provided uncertainty channel, e.g. BetaDistribution from 'effective_count'. WARNING: this method will return the FIRST MATCH from checking distribution types in a non-guaranteed order, so there could be an issue if there are ever distribution types that share an uncertainty channel name. :param uncertainty_channel: name of uncertainty channel to detect a distribution from

Returns: a distribution object

hiv_workflow.lib.analysis.beta_distribution module

class hiv_workflow.lib.analysis.beta_distribution.**BetaDistribution**
Bases: *hiv_workflow.lib.analysis.base_distribution.BaseDistribution*

exception InvalidEffectiveCountException
Bases: Exception

exception InvalidCountChannelException
Bases: Exception

COUNT_CHANNEL = 'effective_count'

UNCERTAINTY_CHANNEL = 'effective_count'

prepare(*dfw*, *channel*, *weight_channel=None*, *additional_keep=None*)
Prepare a DataFrameWrapper and this distribution object for a compare() call together. This includes dataframe verification/data checking, adding additional distribution-specific channels/columns, and trimming the data columns to the minimum needed. Depending on the particular distribution type, additional attributes on self may be set to prepare it in addition to the dfw (e.g. setting self.alpha_channel and self.beta_channel, derived from arg channel on self for BetaDistribution). :param dfw: DataFrameWrapper containing data that will be used in a future compare() call :param channel: data channel/column in dfw that the future compare() call be regarding :param weight_channel: an analyzer weighting channel that must be kept, if specified :param additional_keep: additional columns in the DataFrameWrapper to preserve

Returns: a modified copy of the input DataFrameWrapper

compare(*df*, *reference_channel*, *data_channel*)
Returns a score between -708.3964 and 100 (bad, good) for how well the dataframe (df) simulation data column (data_channel) matches the reference data column (reference_channel). :param df: pandas DataFrame with columns of data to compare :param reference_channel: reference data channel in dataframe :param data_channel: simulation data channel to compare to the reference data channel

Returns: a floating point score measuring the degree of data/reference fit, also known colloquially as 'likelihood'

static construct_beta_channel(*channel*, *type*)

add_percentile_values(*dfw*, *channel*, *p*)
Adds a new data channel to a DataFrameWrapper object that represents a requested probability threshold/value for a specified channel. Useful for creating uncertainty envelopes in plots. :param dfw: DataFrameWrapper with data to construct percentiles and to add percentiles to :param channel: the column in dfw that percentiles will be constructed from/for :param p: the 0-1 percentile level for the given channel to add

Returns: a list containing the new channel name in dfw

add_beta_parameters(*dfw*, *channel*)
Compute and add alpha, beta parameters for a beta distribution to the current self._dataframe object. Distribution is computed for the provided channel (data field), using 'count'. Result is put into new channels/columns named <channel>–Beta-alpha, <channel>–Beta-beta. If both alpha/beta channels already exist in the dataframe, nothing is computed.

Parameters **channel** – The data channel/column to compute the beta distribution for.

Returns a list of the channel-associated alpha and beta parameter channel names.

hiv_workflow.lib.analysis.channel module

class hiv_workflow.lib.analysis.channel.**Channel**(*name*, *type*)
Bases: object

ALLOWED_TYPES = ['fraction', 'count']

exception InvalidChannelType
Bases: Exception

property needs_pop_scaling

hiv_workflow.lib.analysis.condition module

class hiv_workflow.lib.analysis.condition.**Condition**(*stratifier*, *op*, *value*)
Bases: object

apply(*df*)

hiv_workflow.lib.analysis.data_frame_wrapper module

Maybe add xlsx reading, from a defined, similar format to csv

Currently, all files read via .from_directory() are merged into ONE dataframe.

class hiv_workflow.lib.analysis.data_frame_wrapper.**DataFrameWrapper**(*filename=None*, *dataframe=None*, *stratifiers=None*)
Bases: object

exception UnsupportedFileType
Bases: Exception

exception MissingRequiredData
Bases: Exception

exception InconsistentStratification
Bases: Exception

CSV = 'csv'

property channels
Channels are non-stratifier columns :return:

filter(*conditions=None*, *keep_only=None*)
Selects rows from the internal dataframe that satisfy all provided conditions. The stratifiers of the result will exclude current-object stratifiers that contain NaN in the resulting rows.

This method should very rarely if ever be called without a keep_only specified, unless conditions are specified.

Always results in the minimal row/column set satisfying the inputs with no remaining NaN values in the dataframe

Parameters

- **conditions** – an iterator (e.g. list) of tuples/triplets specifying (in order) stratifier, operator, value. e.g. ['min_age', operator.ge, 25] (to select rows where 'min_age' is >= 25)
- **keep_only** – If not None, then is a list of data channels to keep (in addition to stratifiers) after filtering. Rows with any NaN values will be dropped after trimming to these channels.

Returns an object of the same type as the object this method is called on with only selected rows remaining.

merge(*other_dfw*, *index*, *keep_only=None*)
Attempts to merge two DataFrameWrapper objects into one using the provided index list as a multi-index.

Parameters

- **other_dfw** – the DataFrameWrapper object to merge with.
- **index** – a list of columns to merge on. All are required in both DataFrameWrapper objects.
- **keep_only** – a list of columns. Triggers removal of result rows where NaN appears in any of these specified columns. Result will contain these columns AND those from provided index.

Returns A newly created, merged object of the exact type of self and the stratifiers equal to the provided index.

verify_required_items(*needed*, *available=None*)
Standard method for checking if necessary items/channels are available and printing a meaningful error if not :param needed: channels to look for :param available: channel list to look in :return: Nothing

equals(*other_dfw*)

classmethod from_directory(*directory*, *file_type=None*, *stratifiers=None*)

hiv_workflow.lib.analysis.download_analyzer_by_experiment module

class hiv_workflow.lib.analysis.download_analyzer_by_experiment.**DownloadAnalyzerByExperimer**

Bases: idmtools.entities.ianalyzer.IAnalyzer

This analyzer is based on the DownloadAnalyzer and allows the download of files based on a Index and Repetition tags

run_number = 'Run_Number'

sample_tag = '__sample_index__'

directory_for_experiment_and_file(*experiment_id*, *filename*)

per_group(*items*)
Call once before running the apply on the items.

Parameters **items** – Objects with attributes of type ItemId. IDs of one or more higher-level hierarchical objects can be obtained from these IDs in order to perform tasks with them.

Returns None

map(*data*, *item*)
In parallel for each simulation/work item, consume raw data from filenames and emit selected data.

Parameters

- **data** – A dictionary associating filename with content for simulation data.
- **item** – IItem object that the passed data is associated with.

Returns Selected data for the given simulation/work item.

reduce(*all_data*)
Combine the *map()* data for a set of items into an aggregate result.

Parameters **all_data** – A dictionary with entries for the item ID and selected data.

hiv_workflow.lib.analysis.download_analyzer_by_experiment_receipt module

class hiv_workflow.lib.analysis.download_analyzer_by_experiment_receipt.**DownloadAnalyzerByI**

Bases: *hiv_workflow.lib.analysis.download_analyzer_by_experiment.DownloadAnalyzerByExperiment*

This analyzer is based on the DownloadAnalyzer and allows the download of files based on a Index and Repetition tags

hiv_workflow.lib.analysis.gaussian_distribution module

class hiv_workflow.lib.analysis.gaussian_distribution.**GaussianDistribution**
Bases: *hiv_workflow.lib.analysis.base_distribution.BaseDistribution*

exception InvalidUncertaintyException
Bases: Exception

UNCERTAINTY_CHANNEL = 'two_sigma'

prepare(*dfw*, *channel*, *weight_channel=None*, *additional_keep=None*)
Prepare a DataFrameWrapper and this distribution object for a compare() call together. This includes dataframe verification/data checking, adding additional distribution-specific channels/columns, and trimming the data columns to the minimum needed. Depending on the particular distribution type, additional attributes on self may be set to prepare it in addition to the dfw (e.g. setting self.alpha_channel and self.beta_channel, derived from arg channel on self for BetaDistribution). :param dfw: DataFrameWrapper containing data that will be used in a future compare() call :param channel: data channel/column in dfw that the future compare() call be regarding :param weight_channel: an analyzer weighting channel that must be kept, if specified :param additional_keep: additional columns in the DataFrameWrapper to preserve

Returns: a modified copy of the input DataFrameWrapper

static construct_gaussian_channel(*channel*, *type*)

add_percentile_values(*dfw*, *channel*, *p*)
Adds a new data channel to a DataFrameWrapper object that represents a requested probability threshold/value for a specified channel. Useful for creating uncertainty envelopes in plots. :param dfw: DataFrameWrapper with data to construct percentiles and to add percentiles to :param channel: the column in dfw that percentiles will be constructed from/for :param p: the 0-1 percentile level for the given channel to add

Returns: a list containing the new channel name in dfw

compare(*df*, *reference_channel*, *data_channel*)
Returns a score between -708.3964 and 100 (bad, good) for how well the dataframe (df) simulation data column (data_channel) matches the reference data column (reference_channel). :param df: pandas DataFrame with columns of data to compare :param reference_channel: reference data channel in dataframe :param data_channel: simulation data channel to compare to the reference data channel

Returns: a floating point score measuring the degree of data/reference fit, also known colloquially as 'likelihood'

hiv_workflow.lib.analysis.hiv_analyzer module

class hiv_workflow.lib.analysis.hiv_analyzer.**HIVAnalyzer**(*site*, *weight*, *channel*, *scale_population*, *distribution*, *provinciality*, *age_bins='all'*, *debug=False*, *verbose=True*, *output_dir='output'*)
Bases: idmtools_calibra.analyzers.base_calibration_analyzer.BaseCalibrationAnalyzer

exception ProvincialityException
Bases: Exception

exception MissingDataException
Bases: Exception

exception InvalidSiteException
Bases: Exception

SIM_RESULT_CHANNEL = 'Result'

log_float_tiny = -708.3964185322641

AGGREGATED_NODE_MAP = {0: 'All'}

classmethod construct_post_process_filename(*channel*)

map(*data*, *item*)
In parallel for each simulation/work item, consume raw data from filenames and emit selected data.

Parameters

- **data** – A dictionary associating filename with content for simulation data.
- **item** – IItem object that the passed data is associated with.

Returns Selected data for the given simulation/work item.

compare(*sample*, *stratifiers*, *distribution*, *reference_channel*, *data_channel*)

reduce(*all_data*)
Combine the simulation data into a single table for all analyzed simulations.

compute_pop_scaling_factor(*pop_df*)

hiv_workflow.lib.analysis.hiv_calib_site module

class hiv_workflow.lib.analysis.hiv_calib_site.**HIVCalibSite**(***kwargs*)
Bases: idmtools_calibra.calib_site.CalibSite

metadata = {}

DEFAULT_OUTPUT_DIR = 'output'

get_setup_functions()
Derived classes return a list of functions to apply site-specific modifications to the base configuration. These are combined into a single function using the SiteFunctions helper class in the CalibSite constructor.

get_analyzers()
Derived classes return a list of BaseComparisonAnalyzer instances that have been passed a reference to the CalibSite for site-specific analyzer setup.

hiv_workflow.lib.analysis.population_obs module

class hiv_workflow.lib.analysis.population_obs.**PopulationObs**(*filename=None*, *dataframe=None*, *stratifiers=None*)
Bases: *hiv_workflow.lib.analysis.data_frame_wrapper.DataFrameWrapper*

PROVINCIAL = 'Provincial'

NON_PROVINCIAL = 'Non-provincial'

AGGREGATED_NODE = 0

AGGREGATED_PROVINCE = 'All'

WEIGHT_CHANNEL = 'weight'

fix_age_bins()
A method that converts the ‘, ‘ separated AgeBin format: [X, Y) to the new ‘:’ format: [X:Y) for back-compatibility. :return: nothing

get_age_bins()

get_provinces()

get_genders()

get_years()

adjust_years(*exclude_channels=None*)

add_percentile_values(*channel*, *distribution*, *p*)
Computes the inverse beta distribution of ‘value’ at the specified probability threshold. :param channel: the channel/column with beta distribution parameters to compute percentiles with. :param p: probability threshold, float, 0-1 :return: a list of the newly added (single) channel. Adds the column e.g. <channel>–Beta-0.025 (for 2.5000% threshold) for the designated threshold.

find_missing_tuples(*target: object*, *value_column_base: str*, *value_column_target: str = None*) → Optional[List[tuple]]
Finds the missing tuples in the target based on the startifiers and a column containing value. :param target: The target PopulationObs in which to check :param value_column_base: the column containing value in the current object :param value_column_target: the column containing a value in the target

Returns: list of missing tuples for the value_column None if nothing is missing

hiv_workflow.lib.calibration package

Subpackages

hiv_workflow.lib.calibration.algorithms package

Submodules

hiv_workflow.lib.calibration.algorithms.optim_tool module

hiv_workflow.lib.calibration.algorithms.optim_tool.**set_arguments**(*subparsers*, *entry_point*)

hiv_workflow.lib.calibration.algorithms.optim_tool.**initialize**(*args*, *params*, *model*)

hiv_workflow.lib.utils package

Subpackages

hiv_workflow.lib.utils.builders package

Submodules

hiv_workflow.lib.utils.builders.demographics module

hiv_workflow.lib.utils.builders.demographics.**set_risk_assortivities**(*demographics: emodpy_hiv.demographics.HIVD*
risk_assortivity_commercial=Nor
risk_assortivity_informal=None,
risk_assortivity_marital=None,
risk_assortivity_transitory=None
→ None

hiv_workflow.lib.utils.builders.demographics.**set_concurrency_parameters**(*demographics: emodpy_hiv.demographics.*
max_relationships_commer
max_relationships_commer
prob-
a-
bil-
ity_extra_relationship_com
prob-
a-
bil-
ity_extra_relationship_com
max_relationships_commer
max_relationships_commer
prob-
a-
bil-
ity_extra_relationship_com
prob-
a-
bil-
ity_extra_relationship_com
max_relationships_commer
max_relationships_commer
prob-
a-
bil-
ity_extra_relationship_com
prob-
a-
bil-
ity_extra_relationship_com
max_relationships_informa
max_relationships_informa
prob-
a-
bil-
ity_extra_relationship_info
prob-
a-
bil-
ity_extra_relationship_info
max_relationships_informa
max_relationships_informa
prob-
a-
bil-
ity_extra_relationship_info
prob-
a-
bil-
ity_extra_relationship_info
max_relationships_informa
max_relationships_informa
prob-
a-
bil-
ity_extra_relationship_info
prob-
a-

hiv_workflow.lib.utils.builders.demographics.**set_pair_formation_rates**(*demographics: emodpy_hiv.demographics.HIV pair_formation_rate_commerc pair_formation_rate_informal pair_formation_rate_marital= pair_formation_rate_transitor* → None

hiv_workflow.lib.utils.builders.demographics.**set_coital_act_rates**(*demographics: emodpy_hiv.demographics.HIVDem coital_act_rate_commercial=None, coital_act_rate_informal=None, coital_act_rate_marital=None, coital_act_rate_transitory=None*) → None

hiv_workflow.lib.utils.builders.demographics.**set_condom_usage_probabilities**(*demographics: emodpy_hiv.demogra con- dom_usage_min_com con- dom_usage_mid_com con- dom_usage_max_com con- dom_usage_rate_com con- dom_usage_min_info con- dom_usage_mid_info con- dom_usage_max_info con- dom_usage_rate_info con- dom_usage_min_mar con- dom_usage_mid_mar con- dom_usage_max_mar con- dom_usage_rate_mar con- dom_usage_min_tran con- dom_usage_mid_tran con- dom_usage_max_tran con- dom_usage_rate_tran* → None

hiv_workflow.lib.utils.builders.demographics.**set_relationship_durations**(*demographics: emodpy_hiv.demographics.* *commercial_weibull_heterogeneity* *commercial_weibull_scale=None*, *informal_weibull_heterogeneity* *informal_weibull_scale=None*, *marital_weibull_heterogeneity=* *marital_weibull_scale=None*, *transitory_weibull_heterogeneity* *transitory_weibull_scale=None*) → None

hiv_workflow.lib.utils.builders.demographics.**set_initial_risk_distribution**(*demographics: emodpy_hiv.demograph* *initial_risk_distribution_* *float* = *None*) → None

hiv_workflow.lib.utils.builders.general module

exception hiv_workflow.lib.utils.builders.general.**InvalidFunctionArgumentError**
Bases: BaseException

hiv_workflow.lib.utils.builders.general.**get_available_parameters**(*parameter_setting_functions: List[Callable]*, *expected_first_argument: str*) → Dict[Callable, List]

Detects and returns the available custom parameters provided/consumed by a set of functions. The read/detection part of the custom parameter API.

Parameters

- **parameter_setting_functions** – a list of function references that provide/consume custom model parameters
- **expected_first_argument** – the first argument of all provided functions. This will be the object that custom parameters will eventually be set on (not in get_available_parameters). get_available_parameters needs this to know what is NOT a custom parameter in the signature of the provided functions.

Returns A dict, keyed by function references, with values being the list of arguments/custom parameters that the function accepts (e.g. keyword arguments that define available model parameters consumed by the function).

hiv_workflow.lib.utils.builders.general.**set_parameters**(*on: Any*, *parameters_to_set: dict*, *setting_functions: List[Callable]*, *function_arguments: Dict[Callable, List]*) → None

Consumes user-exposed parameters and sets their values on the given object. This function is the write/execution component of the custom parameter API.

Parameters

- **on** – object to set parameter values on (type depends on context)
- **parameters_to_set** – a dict of key/value user-exposed model parameters that need to be set. Not all will necessarily be set during this invocation of set_parameters depending on the behavior of the model input building function(s) calling this method.
- **setting_functions** – a list of function references that consume custom model parameters. All will be called.
- **function_arguments** – This is a dict, keyed by function references, with values being the list of arguments/custom parameters that the function accepts (e.g. keyword arguments that define available model parameters consumed by the function).

Returns None

hiv_workflow.lib.utils.io package

Submodules

hiv_workflow.lib.utils.io.excel module

hiv_workflow.lib.utils.io.excel.**read_block**(*ws*, *range*)

Reads data from the specified range. Result is a list of data by row, each of those being a list (by col). Converts all '' strings to None. :param ws: an openpyxl worksheet object :param range: an openpyxl range object :return: data from the requested 2D worksheet range, in row-major order (list of lists)

hiv_workflow.lib.utils.io.excel.**read_list**(*ws*, *range*)

Read in a linear 1xN or Nx1 range of cells from an excel spreadsheet :param ws: an openpyxl worksheet object :param range: an openpyxl range object :return: data from the requested row/column as a list of items

class hiv_workflow.lib.utils.io.excel.**DefinedName**(*openpyxl_defined_range*)

Bases: object

classmethod load_from_workbook(*wb*)

Submodules

hiv_workflow.lib.utils.analysis module

exception hiv_workflow.lib.utils.analysis.**InvalidDateException**
Bases: Exception

exception hiv_workflow.lib.utils.analysis.**InvalidAgeBinException**
Bases: Exception

exception hiv_workflow.lib.utils.analysis.**InvalidDataframeColumnException**
Bases: Exception

hiv_workflow.lib.utils.analysis.**model_population_in_year**(*year*, *obs_population*, *df*, *low_age=None*, *high_age=None*, *age_bin=None*, *year_col='Year'*, *population_col='Population'*, *verbose=False*)
Computes age [15, 50) (inclusive, exclusive) population from the provided DataFrame in the given year :param year: an Integer. Any offsets are dealt with by this method. Ratio of census population to computed model population is also returned. :param df: data to determine population from :return: model populaton as a numeric, ratio of census/model pop as numeric

hiv_workflow.lib.utils.project_data module

exception hiv_workflow.lib.utils.project_data.**UnsupportedFileFormat**
Bases: Exception

exception hiv_workflow.lib.utils.project_data.**MissingRequiredWorksheet**
Bases: Exception

exception hiv_workflow.lib.utils.project_data.**IncompleteParameterSpecification**
Bases: Exception

exception hiv_workflow.lib.utils.project_data.**IncompleteAnalyzerSpecification**
Bases: Exception

exception hiv_workflow.lib.utils.project_data.**IncompleteDataSpecification**
Bases: Exception

exception hiv_workflow.lib.utils.project_data.**ParameterOutOfRange**
Bases: Exception

exception hiv_workflow.lib.utils.project_data.**InvalidAnalyzerWeight**
Bases: Exception

exception hiv_workflow.lib.utils.project_data.**AnalyzerSheetException**
Bases: Exception

exception hiv_workflow.lib.utils.project_data.**SiteNodeMappingException**
Bases: Exception

exception hiv_workflow.lib.utils.project_data.**ObsMetadataException**
Bases: Exception

hiv_workflow.lib.utils.project_data.**get_ingest_information**(*ingest_filename: str*) → Tuple[dict, *hiv_workflow.lib.analysis.hiv_calib_site.HIVCa*

Returns a dictionary and a HIVCalibSite drawn from parsing the given ingest xlsm file :param ingest_filename: xlsm file to parse for ingest info

Returns: a dict of parsed ingest information and a HIVCalibSite object

hiv_workflow.lib.utils.project_data.**get_sheet_from_workbook**(*wb: openpyxl.workbook.workbook.Workbook*, *sheet_name: str*, *wb_path: str*) → openpyxl.worksheet.worksheet.Worksheet

Obtains a Worksheet from the given Workbook by name :param wb: a Workbook to get a Worksheet from :param sheet_name: name of sheet to retrieve :param wb_path: path of workbook (for error reporting)

Returns: the requested Worksheet

hiv_workflow.lib.utils.project_data.**parse_ingest_data_from_xlsm**(*filename: str*) → Tuple[List[dict], dict, *hiv_workflow.lib.analysis.population_o*List[dict], List[*hiv_workflow.lib.analysis.channel.*

Parses an ingest xlsm file into various types of information :param filename: xlsm file to parse for ingest info

Returns: parsed ingest information

hiv_workflow.lib.utils.runtime module

hiv_workflow.lib.utils.runtime.**add_post_channel_config_as_asset**(*task: idmtools.entities.itask.ITask*, *channels: List[*hiv_workflow.lib.analysis.channel.*reference_data:* hiv_workflow.lib.analysis.population_o*site_info: dict*) → None

Construct a post_channel_config.json file to configure the EMOD post-processor for HIV analyzer input and ensure it is added to Task assets

Parameters

- **task** – Task object to add file as an asset to
- **channels** – list of Channels, model data channels to post-process
- **reference_data** – reference data from an ingest form for determining which provinces/ages/etc to post-process model results at.
- **site_info** – information related to scaling of simulations back to reference population

Returns None

hiv_workflow.lib.utils.runtime.**compute_num_cores**(*max_memory_mb: int*) → int

Computes the number of cores to request for a simulation based on an assumption of one core per 8GB of requested memory for EMOD.

Parameters **max_memory_mb** – simulation max memory, in MB

Returns Number of cores to request

hiv_workflow.lib.utils.runtime.**set_python_processing**(*task: idmtools.entities.itask.ITask*, *pre: str = None*, *mid: str = None*, *post: str = None*) → idmtools.entities.itask.ITask

Adds EMOD python pre/in/post processing file(s) to the given task as assets and makes sure embedded python is turned off if none are given (it currently defaults to on).

Parameters

- **task** – the Task object to potentially add pre/in/post processing files to
- **pre** – Defines the pre-processor to use. If 'standard', use the in-code standard processor, if a path, the path to a custom pre-processor, if None, no pre-processor will be used.
- **mid** – Defines the in-processor to use. If 'standard', use the in-code standard processor, if a path, the path to a custom in-processor, if None, no in-processor will be used.
- **post** – Defines the post-processor to use. If 'standard', use the in-code standard processor, if a path, the path to a custom post-processor, if None, no post-processor will be used.

Returns The provided task object, modified

hiv_workflow.lib.utils.runtime.**add_ingest_form_to_assets**(*task: idmtools.entities.itask.ITask*, *path: str*) → idmtools.entities.itask.ITask

Simply adds the ingest form specified as an asset to the provided task for logging purposes

Parameters

- **task** – the Task object to add the ingest form to
- **path** – path of the ingest form to add

Returns The provided task object, modified

hiv_workflow.lib.utils.runtime.**map_sample_to_model_input**(*simulation: idmtools.entities.simulation.Simulation*, *sample: dict*, *config_builder: Callable = None*, *campaign_builder: Callable = None*, *demographics_builder: Callable = None*, *random_run_number: bool = True*) → dict

This method builds config, campaign, and/or demographics objects and sets them on the provide simulation object. It consumes a sample, which is a dict of parameter key/values that are used as overrides to the default behavior of the provided builders.

Parameters

- **simulation** – simulation object to add built config/campaign/demographics objects to
- **sample** – dict of parameter names/values overrides to use during config/campaign/demographics building
- **config_builder** – reference to a function that builds an EMOD config object. None means do not build a config object.

- **campaign_builder** – reference to a function that builds an EMOD campaign object. None means do not build a campaign object.
- **demographics_builder** – reference to a function that builds an EMOD demographics object. None means do not build a demographics object.
- **random_run_number** – if True, run numbers will be randomly assigned, otherwise any pre-existing run number will be used

Returns A dict of simulation tag names/values

hiv_workflow.lib.utils.runtime.**constrain_sample**(*sample: dict*, *custom_sample_constrainer: Callable*) → dict

Calls the provided constraint function on the given sample. Its purpose is to ensure various logical properties in the sample.

Parameters

- **sample** – sample to check for logical constraints
- **custom_sample_constrainer** – function that accepts a sample and modifies it based on internal constraint logic

Returns The provide sample, modified

hiv_workflow.lib.utils.runtime.**initialize_EMODTask**(*executable_path: str*, *schema_path: str*, *config_builder: Callable = None*, *campaign_builder: Callable = None*, *demographics_builder: Callable = None*, *demographics_paths: List[str] = None*, *python_processing_setter: Callable = None*, *channels: List[hiv_workflow.lib.analysis.channel.Channel] = None*, *reference: hiv_workflow.lib.analysis.population_obs.PopulationObs = None*, *site_info: dict = None*) → emodpy.emod_task.EMODTask

Creates an EMODTask object with specified executable, builders, etc set on it.

Parameters

- **executable_path** – path to where the model binary is/will be
- **schema_path** – path to schema matching model binary
- **config_builder** – reference to a function that builds an EMOD config object. None means do not build a config object.
- **campaign_builder** – reference to a function that builds an EMOD campaign object. None means do not build a campaign object.
- **demographics_builder** – reference to a function that builds an EMOD demographics object. None means do not build a demographics object. Mutually exclusive with demographics_paths.
- **demographics_paths** – list of paths to demographics files to use. Mutually exclusive with demographics_builder.

- **python_processing_setter** – reference to a function that sets EMOD python pre/in/post processing on an EMODTask object and adds any appropriate assets.
- **channels** – list of channels for configuring HIV post-processing for analysis
- **reference** – reference data object for configuring HIV post-processing for analysis
- **site_info** – information related to scaling of simulations back to reference population, for configuring HIV post-procesing for analysis

Returns A configured EMODTask object

hiv_workflow.lib.utils.runtime.**load_model**(*model_name: str*, *model_root: str = 'models'*) → module

Loads a model directory by name

Parameters

- **model_name** – name of model (directory) to load
- **model_root** – directory containing model to load

Returns The loaded model

hiv_workflow.lib.utils.runtime.**available_algorithms**() → List[str]

Returns A list of calibration algorithms available for use by name

hiv_workflow.lib.utils.runtime.**load_algorithm**(*algorithm_name: str*, *algorithm_root: str = 'hiv_workflow.lib.calibration.algorithms'*) → module

Loads a calibration algorithm binding code by name relative to a code path

Parameters

- **algorithm_name** – the name of the algorithm to load binding code for.
- **algorithm_root** – the path to a module containing algorithm bindings to load

Returns The module of binding code for the specified calibration algorithm

hiv_workflow.lib.utils.runtime.**detect_duplicate_items_in**(*items: Iterable*) → List

Simple function that detects and returns the duplicates in an provide iterable.

Parameters **items** – a collection of items to search for duplicates

Returns A list of duplicated items from the provided list

hiv_workflow.lib.utils.wrappers module

hiv_workflow.lib.utils.wrappers.**generate_config_builder_wrapper**(*config_builder: Callable*, *schema_path: str*, *kwargs: dict*, *max_memory_mb: int = None*, *demographics_paths: List[str] = None*, *dry_run: bool = False*) → Callable

hiv_workflow.lib.utils.wrappers.**generate_campaign_builder_wrapper**(*campaign_builder: Callable*, *schema_path: str*, *kwargs: dict*, *dry_run: bool = False*) → Callable

hiv_workflow.lib.utils.wrappers.**generate_demographics_builder_wrapper**(*demographics_builder: Callable*, *manifest: module*, *kwargs: dict*, *dry_run: bool = False*) → Callable

hiv_workflow.lib.utils.wrappers.**generate_map_sample_to_model_input_wrapper**(*config_builder: Callable*, *campaign_builder: Callable*, *demographics_builder: Callable*, *random_run_number: bool*) → idmtools_calibra.calib_ma

hiv_workflow.lib.utils.wrappers.**constrain_sample_wrapper**(*custom_sample_constrainer: Callable*) → Callable

hiv_workflow.lib.utils.wrappers.**generate_python_processing_wrapper**(*pre: str = None*, *mid: str = None*, *post: str = None*) → Callable

hiv_workflow.scripts package

Submodules

hiv_workflow.scripts.available_parameters module

hiv_workflow.scripts.calibrate module

hiv_workflow.scripts.download module

hiv_workflow.scripts.dtk_post_process module

hiv_workflow.scripts.resample module

hiv_workflow.scripts.run module

PYTHON MODULE INDEX

h

INDEX

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W