Mesa Data Collection Module

DataCollector is meant to provide a simple, standard way to collect data generated by a Mesa model. It collects three types of data: model-level data, agent-level data, and tables.

A DataCollector is instantiated with two dictionaries of reporter names and associated variable names or functions for each, one for model-level data and one for agent-level data; a third dictionary provides table names and columns. Variable names are converted into functions which retrieve attributes of that name.

When the collect() method is called, each model-level function is called, with the model as the argument, and the results associated with the relevant variable. Then the agent-level functions are called on each agent in the model scheduler.

Additionally, other objects can write directly to tables by passing in an appropriate dictionary object for a table row.

The DataCollector then stores the data it collects in dictionaries:

model_vars maps each reporter to a list of its values

tables maps each table to a dictionary, with each column as a key with a list as its value.

_agent_records maps each model step to a list of each agents id and its values.

Finally, DataCollector can create a pandas DataFrame from each collection.

The default DataCollector here makes several assumptions:

The model has a schedule object called ‘schedule’

The schedule has an agent list called agents

For collecting agent-level variables, agents must have a unique_id

class **DataCollector(*model_reporters=None*, *agent_reporters=None*, *tables=None*)** [source]

Class for collecting data generated by a Mesa model.

A DataCollector is instantiated with dictionaries of names of model- and agent-level variables to collect, associated with attribute names or functions which actually collect them.

When the collect(…) method is called, it collects these attributes and executes these functions one by one and stores the results.

Instantiate a DataCollector with lists of model and agent reporters. Both model_reporters and agent_reporters accept a dictionary mapping a variable name to either an attribute name, or a method. For example, if there was only one model-level reporter for number of agents, it might look like:

{“agent_count”: lambda m: m.schedule.get_agent_count() }

If there was only one agent-level reporter (e.g. the agent’s energy), it might look like this:

{“energy”: “energy”}

or like this:

{“energy”: lambda a: a.energy}

The tables arg accepts a dictionary mapping names of tables to lists of columns. For example, if we want to allow agents to write their age when they are destroyed (to keep track of lifespans), it might look like:

{“Lifespan”: [“unique_id”, “age”]}

Args:

model_reporters: Dictionary of reporter names and attributes/funcs agent_reporters: Dictionary of reporter names and attributes/funcs. tables: Dictionary of table names to lists of column names.

Notes:

If you want to pickle your model you must not use lambda functions. If your model includes a large number of agents, you should *only* use attribute names for the agent reporter, it will be much faster.

Model reporters can take four types of arguments: lambda like above: {“agent_count”: lambda m: m.schedule.get_agent_count() } method of a class/instance: {“agent_count”: self.get_agent_count} # self here is a class instance {“agent_count”: Model.get_agent_count} # Model here is a class class attributes of a model {“model_attribute”: “model_attribute”} functions with parameters that have placed in a list {“Model_Function”:[function, [param_1, param_2]]}

collect(*model*) **[source]**

Collect all the data for the given model object.

add_table_row(*table_name*, *row*, *ignore_missing=False*) [source]

Add a row dictionary to a specific table.

Args:

table_name: Name of the table to append a row to. row: A dictionary of the form {column_name: value…} ignore_missing: If True, fill any missing columns with Nones;

if False, throw an error if any columns are missing

get_model_vars_dataframe() [source]

Create a pandas DataFrame from the model variables.

The DataFrame has one column for each model variable, and the index is (implicitly) the model tick.

get_agent_vars_dataframe() [source]

Create a pandas DataFrame from the agent variables.

The DataFrame has one column for each variable, with two additional columns for tick and agent_id.

get_table_dataframe(*table_name*) [source]

Create a pandas DataFrame from a particular table.

Args:

table_name: The name of the table to convert.