

---

# METER

*Release 0.3.3*

**Malcolm Risk**

**Mar 06, 2021**



# CONTENTS

<b>1</b>	<b>Guide</b>	<b>3</b>
1.1	Should I use METER? . . . . .	3
1.2	Installation . . . . .	4
1.3	Constructing Multi-State Life Tables . . . . .	4
1.4	Generating Survival Estimates . . . . .	6
1.5	Graphical Summaries . . . . .	9
<b>2</b>	<b>General</b>	<b>11</b>
2.1	License . . . . .	11
2.2	Need Help? . . . . .	13
<b>3</b>	<b>Indices and tables</b>	<b>15</b>
	<b>Index</b>	<b>17</b>



METER (Multi-state Estimates for Time-to-Event Research) is a Python package for multi-state modelling of time-to-event data. It contains functions for building multi-state life tables from discrete-time data that can be used to generate point estimates and confidence intervals for life expectancies. METER also provides functions to produce plots of transition probabilities and Kaplan-Meier plots of survival data.



## 1.1 Should I use METER?

METER is specifically designed for discrete-time, time-inhomogenous, and acyclic multi-state markov models. To go through each of these elements individually:

- The main assumption of a **markov model** is that the past states of a process do not affect transition probabilities going forward. If you have an application where the previous states of a process are likely to have an effect (if you are studying radiation exposure, for example) this assumption will not hold and you should consider using a semi-markov model.
- 
- **Discrete-time** means that observations are taken at regular time intervals that can be indexed by natural numbers. Typically this means that the data is indexed by days or years. This assumption will not hold for data with inconsistent intervals between observations, which would be typical for clinical data on cancer relapse and many types of survey data.
- 
- **Acyclic** means that states have a defined order and it is not possible to move backwards to a preceding state. For the time being, METER does not support cyclic models, although this is high on the list of planned improvements.
- 
- **Time-inhomogenous** means that time from study entry affects the probability of transitions between states. This is typical for data that stretches over long periods of time, where subject age may be a relevant factor. For example, an individual diagnosed with type 2 diabetes at age 20 is going to have a much better short-term survival prognosis than an 80-year-old with the same diagnosis.

You can still use METER if you believe that your process is time-homogenous, but you might have simpler options available. If you want to model a continuous time process, I suggest using the R package [MSM](#).

If you haven't used Python before I would recommend going through a tutorial or two and familiarizing yourself with basic datatypes, but you will not need any advanced knowledge of the language to use METER. You should also make sure that you know how to import .xls and .csv files as pandas dataframes. If you want to do anything beyond the core functionality of the package a decent background in [pandas](#) will be very useful.

## 1.2 Installation

Run the following in your command line or python console:

```
pip install METER
```

For a look at past versions or to download files directly, visit [PyPI](#).

## 1.3 Constructing Multi-State Life Tables

The first step to using METER is to have your data in the correct format. The data should have one subject per row, and have columns containing each of the transition times. Transition times can be specified as calendar dates or time from study entry, as the `wide_format` function will convert everything into the latter format.

**To obtain the transition matrices of the multi-state model you should do the following:**

1. Use `wide_format()` to obtain the data in wide format.
2. Use `atrisk_and_transitions()` to obtain the risk sets and number of transitions made at each time point.
3. Use `transitionprobs_and_samplesizes()` to obtain the transition matrices for each time point, and the associated sample sizes.

If you don't need the transition probabilities directly, and just want associated life expectancies or other survival estimates, you can just do step one and then skip forward to the section on generating estimates.

---

**Note:** You should still be running `wide_format` even if your data already has transition times specified as time from study entry. This is because the `wide_format` function creates status columns that indicate whether each transition ever occurred for a given subject, and ensures that there is only one transition per year.

---

METER also allows you to censor data at a given state. This is often useful for reasons mentioned in [Life Expectancies](#). If you want to censor your data, you should do this immediately after step 1 above, using `censor()`.

`METER.wrangler.wide_format(data, transition_names, exit)`

Take data with transitions as calendar dates and expand for life table construction. If multiple transitions occur at the same time (i.e. in the same year or day) then only the final transition is counted as occurring.

### Parameters

- **data** (*pandas dataframe*) – the data, with dates of all transitions, with one subject per line
- **transition\_names** (*list*) – a list of the names of the columns that contain the transition times
- **exit** (*string*) – the name of the column indicating final follow-up date for each subject

### Returns

a dataframe with additional columns for:

- transitions as time from study entry
- transition status columns indicating whether such a transition ever occurred
- a column `final_age`, for age at study close-out

**Return type** pandas dataframe



`METER.table.atrisk_and_transitions(data, transition_names, states)`

Get the number of individuals in a given state at a each time, and number of transitions at each time.

#### Parameters

- **data** (*pandas dataframe*) – the data in wide format as generated by `wide_format()`
- **transition\_names** (*list*) – a list of the names of the columns that contain the transition times
- **states** (*list*) – the names of the states in the model, the entry into each of which will correspond to the columns in `transition_names`

#### Returns

a two element list with:

- first element a dataframe containing the risk sets for each time point
- second element a dataframe containing the number of transitions of each type at each time point

**Return type** list

`METER.table.transitionprobs_and_samplesizes(riskdf, transdf, states)`

Obtain transition matrices governing transitions from each time point, and associated sample sizes.

#### Parameters

- **riskdf** (*pandas dataframe*) – a dataframe with the number of individuals in each state, indexed at each time point. This is the first output from `atrisk_and_transitions()`
- **transdf** (*pandas dataframe*) – a dataframe containing the number of transitions of each type at each time point. The second output from `atrisk_and_transitions()`
- **states** (*list*) – the names of the states in the model

#### Returns

a two element list with:

- first element a list containing the transition matrix (as a numpy array) for each time point
- second element a list containing a matrix of the associated sample sizes

**Return type** list

`METER.wrangler.censor(data, transition, transition_names)`

censor all individuals in a dataframe at a given state

#### Parameters

- **data** (*pandas dataframe*) – the dataframe in wide format as created by `wide_format()`
- **transition** (*string*) – the column name in the original dataset that represents transitions into the state you want to censor at
- **transition\_names** (*list*) – a list of the names of the columns that contain the transition times

**Returns** a new dataframe where all individuals have been censored at the desired state

**Return type** pandas dataframe

## 1.4 Generating Survival Estimates

### 1.4.1 Survival Predictions

First, make sure that you have generated the *transition matrices*.

Future state probabilities for any population of individuals from an initial time can be generated using `survivorship_vector()`. The most common application is likely to be generating future state probabilities for one individual, which can be obtained by specifying a radix with a 1 in the position of the state you want to predict from. At this point you can also generate a life expectancy for an individual starting in any state at any time from study entry using `life_expectancy()`, although it is possible to use `ensorLE()` to obtain this directly from the data in wide format.

---

**Note:** It is implicitly assumed that the final state in your model is death, but this will also work for other outcomes that are the terminal state of a study. For example, if your study outcome is the age of reaching menopause then the life expectancy function will still give you a valid estimate.

---

`METER.table.survivorship_vector(transmat, radix, initial_time, states)`

Predict the expected proportion of a population being in a given state at each time point based on some initial time and number of individuals currently in each state. To get probabilities for one individual, simply proceed for a 1-individual population.

#### Parameters

- **transmat** (*list*) – a list of numpy matrices that represent the transition probabilities at each time point. This is the first output from `transitionprobs_and_samplesizes()`.
- **states** (*list*) – the names of the states in the model
- **radix** (*numpy array*) – an initial condition for the number of subjects in each state we want to model, specified as a numpy vector. For example, if we wanted the probabilities of being in each state for 1 individual starting at the first state in a 6-state model the radix would be generated by `np.array([[1],[0],[0],[0],[0],[0]], dtype=float)`.
- **initial\_time** (*int*) – the initial time to model the survivorship outcomes from

**Returns** a dataframe with the expected proportion of the population in each state at each time point past the initial time given

**Return type** pandas dataframe

`METER.table.life_expectancy(transmat, initial_state, initial_time, states)`

Predict the life expectancy of an individual based on current state and age

#### Parameters

- **transmat** (*list*) – a list of numpy matrices that represent the transition probabilities at each age. This is the first output from `transitionprobs_and_samplesizes()`.
- **states** (*list*) – the names of the states in the model.
- **initial\_state** (*string*) – the initial state to model the survivorship outcomes from.
- **initial\_age** (*int*) – the initial time to model the survivorship outcomes from

**Returns** the life expectancy of an individual given the parameters specified

**Return type** float

## 1.4.2 Life Expectancies

For the methods in this section, it is not necessary to have obtained the transition matrices. However, make sure that you have used `wide_format()` to get the data in the correct format.

METER permits 4 different ways of characterizing an individual when computing life expectancies:

- The **initial state** is the state that the individual starts in, and is required whenever you ask METER to compute life expectancy point estimates or confidence intervals.
- The **initial time** is the time point that you are predicting from. METER always gives life expectancy from this point forwards, so if you want absolute life expectancy estimates from study entry you should add this on after you've obtained the estimates.
- A **sensor state** is a state that you want to restrict the individual from moving beyond. This is useful for applications where you want to assess the life expectancy effect of attaining a certain state (eg. an award or military rank) and want to prevent your estimates for the control group from being affected by potential future transitions to that state.
- Finally, you can specify any arbitrary set of **covariate conditions** for subgroup analysis. For the time being these need to be categorical covariates, although this is on the list of improvements for future versions.

Point estimates of life expectancy can be obtained using `sensorLE()` directly from the data in wide format. As mentioned above, the initial state is a mandatory input. Unless the other conditions mentioned are specified, METER will assume that you want estimates from study entry (time point 0), with no censoring, and with no covariate restrictions.

To obtain confidence intervals by non-parametric bootstrap, you can use `bootstrapLE()` to get a dataframe of bootstrap runs, and then use `summary_results()` to obtain the confidence intervals and point estimates. METER allows you to run the bootstrap for any arbitrary set of groups, each defined by different initial states, sensor states, and covariate restrictions. The initial time must be constant over each of these groups, because METER also provides confidence intervals and point estimates for the difference between these groups, and those measures only make sense if each group begins at the same time point from study entry.

METER is fast. 1000 bootstraps on a dataset of 5000 individuals with less than 10 states in the model for less than 5 groups should take under 15 minutes. If you want a log of how far your bootstrap has progressed printed to the console set `loud=TRUE`, and you will know whether you have time to go get a coffee.

**Note:** If you are comparing life expectancies for a number of different groups, I highly recommend using the `group_names` input to `bootstrapLE`. By default the groups will be named based on the initial states and these names may not be unique, which could cause a great deal of confusion.

```
METER.table.sensorLE(data, transition_names, states, initial_state, initial_time=0,
                      sensor_state='default', conditions=None)
```

Get the life expectancy of an individual based on initial state, initial time, and any sensor states or covariate conditions.

### Parameters

- **data** (*pandas dataframe*) – the data in wide format as generated by `wide_format()`
- **transition\_names** (*list*) – a list of the names of the columns that contain the transition times

- **states** (*list*) – the names of the states in the model
- **initial\_state** (*string*) – the initial state that you want to estimate life expectancy from
- **initial\_time** (*int*) – optional input if you want to estimate life expectancy after a given time (by default 0)
- **censor\_state** (*string*) – a particular state that you want to restrict movement beyond (by default none)
- **conditions** (*dictionary*) – a set of conditions you want the group to be subject to (by default none). ex. {'Smoking': 'Yes', 'Race': 'White'}

**Returns** the life expectancy of an individual starting in the initial state given the conditions provided

**Return type** float

```
METER.summaries.bootstrapLE(data, transition_names, states, initial_states, n=1000, initial_time=0, censor_states='default', group_names='default', conditions='default', loud=False)
```

Run a bootstrap on the life expectancy for a given set of groups

#### Parameters

- **data** (*pandas dataframe*) – the data in wide format as generated by `wide_format()`
- **transition\_names** (*list*) – a list of the names of the columns that contain the transition times
- **states** (*list*) – the names of the states in the model
- **initial\_states** (*list*) – a list of initial states to estimate from
- **n** (*int*) – the number of bootstraps to run, by default 1000.
- **initial\_time** (*int*) – to estimate life expectancy after a given time (by default 0)
- **censor\_states** (*list*) – the states you want each group's life expectancy to be censored at (by default no censoring) if provided this list must be the same length as `initial_states`
- **group\_names** (*list*) – what the groups (whose structure is defined both by the initial states and censor states given) are to be called. by default this is the same as the initial states. if provided this list must be the same length as `initial_states`
- **conditions** (*list*) – a list of dictionaries of conditions you want each group to be subject to (by default none). ex. [{'Race': 'White', 'Smoking': 'Yes'}, {'Race': 'Black', 'Smoking': 'No'}]
- **loud** (*bool*) – by default this is false. If it is set to true a small summary of the results of each bootstrap as well as the best estimate calculated initially are printed to the console.

**Returns** a dataframe containing the results of each run of the bootstrap. Each row will include that bootstrap life expectancy for each group as well as each of the possible group differences.

**Return type** pandas dataframe

```
METER.summaries.summary_results(bootstrap, confidence_level=0.95)
```

Summarize the results of a bootstrap.

#### Parameters

- **bootstrap** (*pandas dataframe*) – the bootstrap dataframe generated by `bootstrapLE()`

- **confidence\_level** (*float*) – the confidence level which you want to generate confidence intervals for

**Returns** A dataframe summarizing the point estimates and confidence intervals for each quantity.

**Return type** pandas dataframe

## 1.5 Graphical Summaries

If you have obtained future state probabilities for an individual using `survivorship_vector()`, you can obtain a nice stacked area plot of state probabilities using `survivorship_graph()`. The aesthetic value of these plots is heavily dependent on your choice of colours, so you may want to play around with that a bit.

`METER.plots.survivorship_graph(survdf, states, colors, order='default')`

Stacked area chart of state probabilities for an individual.

### Parameters

- **survdf** (*pandas dataframe*) – a dataframe containing the given probabilities of being in each state at each age for some initial condition. This is precisely the output of `survivorship_vector()`.
- **states** (*list*) – a list of the states in the model
- **colors** (*list*) – a list of colors accepted by matplotlib (as strings) that matches the number of states
- **order** (*list*) – the order you want the states to be from top to bottom in the graph. The default is reverse order, which is fairly visually appealing. If an input is passed it should be something like `[0,1,2,3]` with 0 representing the first state, 1 the second state, etc.

**Returns** a graph of state proportions for some initial conditions

**Return type** plot



**GENERAL**

## 2.1 License

Creative Commons Legal Code

CC0 1.0 Universal

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS DOCUMENT DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN “AS-IS” BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE USE OF THIS DOCUMENT OR THE INFORMATION OR WORKS PROVIDED HEREUNDER, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM THE USE OF THIS DOCUMENT OR THE INFORMATION OR WORKS PROVIDED HEREUNDER.

### Statement of Purpose

The laws of most jurisdictions throughout the world automatically confer exclusive Copyright and Related Rights (defined below) upon the creator and subsequent owner(s) (each and all, an “owner”) of an original work of authorship and/or a database (each, a “Work”).

Certain owners wish to permanently relinquish those rights to a Work for the purpose of contributing to a commons of creative, cultural and scientific works (“Commons”) that the public can reliably and without fear of later claims of infringement build upon, modify, incorporate in other works, reuse and redistribute as freely as possible in any form whatsoever and for any purposes, including without limitation commercial purposes. These owners may contribute to the Commons to promote the ideal of a free culture and the further production of creative, cultural and scientific works, or to gain reputation or greater distribution for their Work in part through the use and efforts of others.

For these and/or other purposes and motivations, and without any expectation of additional consideration or compensation, the person associating CC0 with a Work (the “Affirmer”), to the extent that he or she is an owner of Copyright and Related Rights in the Work, voluntarily elects to apply CC0 to the Work and publicly distribute the Work under its terms, with knowledge of his or her Copyright and Related Rights in the Work and the meaning and intended legal effect of CC0 on those rights.

1. Copyright and Related Rights. A Work made available under CC0 may be protected by copyright and related or neighboring rights (“Copyright and Related Rights”). Copyright and Related Rights include, but are not limited to, the following:

- (i). the right to reproduce, adapt, distribute, perform, display, communicate, and translate a Work;
- (ii). moral rights retained by the original author(s) and/or performer(s);
- (iii). publicity and privacy rights pertaining to a person’s image or likeness depicted in a Work;
- (iv). rights protecting against unfair competition in regards to a Work, subject to the limitations in paragraph 4(a), below;

- (v). rights protecting the extraction, dissemination, use and reuse of data in a Work;
- (vi). database rights (such as those arising under Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, and under any national implementation thereof, including any amended or successor version of such directive); and
- (vii). other similar, equivalent or corresponding rights throughout the world based on applicable law or treaty, and any national implementations thereof.

2. Waiver. To the greatest extent permitted by, but not in contravention of, applicable law, Affirmer hereby overtly, fully, permanently, irrevocably and unconditionally waives, abandons, and surrenders all of Affirmer's Copyright and Related Rights and associated claims and causes of action, whether now known or unknown (including existing as well as future claims and causes of action), in the Work (i) in all territories worldwide, (ii) for the maximum duration provided by applicable law or treaty (including future time extensions), (iii) in any current or future medium and for any number of copies, and (iv) for any purpose whatsoever, including without limitation commercial, advertising or promotional purposes (the "Waiver"). Affirmer makes the Waiver for the benefit of each member of the public at large and to the detriment of Affirmer's heirs and successors, fully intending that such Waiver shall not be subject to revocation, rescission, cancellation, termination, or any other legal or equitable action to disrupt the quiet enjoyment of the Work by the public as contemplated by Affirmer's express Statement of Purpose.

3. Public License Fallback. Should any part of the Waiver for any reason be judged legally invalid or ineffective under applicable law, then the Waiver shall be preserved to the maximum extent permitted taking into account Affirmer's express Statement of Purpose. In addition, to the extent the Waiver is so judged Affirmer hereby grants to each affected person a royalty-free, non transferable, non sublicensable, non exclusive, irrevocable and unconditional license to exercise Affirmer's Copyright and Related Rights in the Work (i) in all territories worldwide, (ii) for the maximum duration provided by applicable law or treaty (including future time extensions), (iii) in any current or future medium and for any number of copies, and (iv) for any purpose whatsoever, including without limitation commercial, advertising or promotional purposes (the "License"). The License shall be deemed effective as of the date CC0 was applied by Affirmer to the Work. Should any part of the License for any reason be judged legally invalid or ineffective under applicable law, such partial invalidity or ineffectiveness shall not invalidate the remainder of the License, and in such case Affirmer hereby affirms that he or she will not (i) exercise any of his or her remaining Copyright and Related Rights in the Work or (ii) assert any associated claims and causes of action with respect to the Work, in either case contrary to Affirmer's express Statement of Purpose.

#### 4. Limitations and Disclaimers.

- a. No trademark or patent rights held by Affirmer are waived, abandoned, surrendered, licensed or otherwise affected by this document.
- b. Affirmer offers the Work as-is and makes no representations or warranties of any kind concerning the Work, express, implied, statutory or otherwise, including without limitation warranties of title, merchantability, fitness for a particular purpose, non infringement, or the absence of latent or other defects, accuracy, or the present or absence of errors, whether or not discoverable, all to the greatest extent permissible under applicable law.
- c. Affirmer disclaims responsibility for clearing rights of other persons that may apply to the Work or any use thereof, including without limitation any person's Copyright and Related Rights in the Work. Further, Affirmer disclaims responsibility for obtaining any necessary consents, permissions or other rights required for any use of the Work.
- d. Affirmer understands and acknowledges that Creative Commons is not a party to this document and has no duty or obligation with respect to this CC0 or use of the Work.



### 2.1.1 Contact

Questions? Please contact [mrisk435@gmail.com](mailto:mrisk435@gmail.com)

## 2.2 Need Help?

Please contact [mrisk435@gmail.com](mailto:mrisk435@gmail.com)



## INDICES AND TABLES

- `genindex`



## INDEX

### A

`atrisk_and_transitions()` (in module *METER.table*), 4

### B

`bootstrapLE()` (in module *METER.summaries*), 8

### C

`censor()` (in module *METER.wrangler*), 5

`censorLE()` (in module *METER.table*), 7

### L

`life_expectancy()` (in module *METER.table*), 6

### S

`summary_results()` (in module *METER.summaries*), 8

`survivorship_graph()` (in module *METER.plots*), 9

`survivorship_vector()` (in module *METER.table*), 6

### T

`transitionprobs_and_samplesizes()` (in module *METER.table*), 5

### W

`wide_format()` (in module *METER.wrangler*), 4