MiniMarkov

MiniMarkov is a library of routines designed to model state based Markov models. It contains two classes, markov and result_set. This tutorial demonstrates the use of this library.

Consider a simplified process of leaving a hotel room and exiting the hotel. The following states may be considered to occur:

- In room
- Hallway
- Lobby
- Hotel door

You then need to set a series of probabilities of moving from state to state. This is called the Transitions Table and is listed below:

To From

	In Room	Hallway	Lobby	Hotel Door
In Room	0.8	0.25	0	0
Hallway	0.2	0.5	0.2	0
Lobby	0	0..25	0.5	0
Hotel Door	0	0	0.3	1

In this table the probability of transitioning between states is laid out. For example, we say that the likelihood of staying in the room is 0.8, and 0.2 chance of going to the Hallway. From the Hallway there is a 0.1 chance of going back into the room (forgot a cell phone, e.g.), 0.6 of staying in the Hallway, and 0.3 that you reach the Lobby. Once in the Lobby there is a 0.2 chance of going back into the Hallway, 0.2 staying in the Lobby, and 0.6 of exiting through the Hotel Door. Once you reach the Hotel Door, this is a “Terminal” state for the simulation and you go no further.

So, how do you program this into MiniMarkov and what data can be extracted? Start by importing Markov and, optionally, a progress bar module to see your progress and a callback function to implement it. Then define the transition table as a list of lists. Use the mini_markov build_df function to convert the transition table to a dataframe, then use the run function to perform the analysis. You get back a result set that the results_range function will give you the minimum, maximum, mean, median, and standard deviation of the results from the series of trials.

```
    Import markov
    import progressbar
    p = progressbar.ProgressBar(maxval=1000).start()
    def callbac(progress):
        p.update(progress)
        return
```

The above code imports the optional progress bar and sets up the callback function.

```
    tmatrix = [[0.8,0.2,0,0],[0.25,0.5,0.25,0],[0,0.2,0.5,0.3],[0,0,0,1]]
    names = ['Room', 'Hall', 'Lobby', 'Exit']
    m = mini_markov()
    df = m.build_df(names, tmatrix)
```

The above code defines the transition matrix as a list of lists as well as the list of names for the states. You then instantiate mini_markov and use the build_df function to create the formatted dataframe. Note that the markov class imports pandas and numpy so you do not have to.

```
    rset = m.run(df,trials=1000,epochs=100,startstate=0,seed=42, log='test.csv',
logtype = 'csv',progress=callbac)
    p.finish()
```

You then use the run function to run the simulation using the following parameters:

1. The dataframe containing the transition table and names
2. trials=x where x is the number of trials in your simulation, default is 100
3. epochs=y where y is the maximum number of steps per trial, default is 100
4. startstate=z where z is (numerically) the starting state, default is 0
5. log=fn where fn is a filename, default is None. This allows you to see the raw data from the simulation. Log files can be quite long.
6. logtype=type Default is none, options are text or csv
7. seed=s This is the random number seed for the randomization functions. Default is -1 which is no random seed being set.
8. progress=cb Default is none, this is where you set the callback function to give you a progress bar.

In this example, rset is of the result_set() subclass containing the trial and epoch data. There are several result_set() functions that you can use to examine aggregate data.

```
print(f'Time spent in each state:\n{rset.counts()[0]} \n{rset.counts()[1]}')
print(f'Average time through system: {rset.mean_time()}')
print(rset.results_range())
values = [20,35,30,10]
print(rset.value(values))
```

The counts() function returns state information for each state as a tuple. The first value is the number of epochs in each state, and the second is the percentage. Note that both are pandas dataframes. The mean_time() function returns the mean time for going through the system. If there are no terminal states this may be the maximum epochs value.. results_range() gives the statistics on the result_set including minimum, maximum, mean, median, and standard deviation for the number of epochs spent in each state.

A word about value_array and values. You have the option of giving each state a value and then getting statistics based on those values as well. Declare a list of values, one for each state, then call result_set.value(value_array) which returns a dataframe containing those same values when calculated on the modified results array.. Value_array is an optional parameter in results_range().

Results should look like this:

```
Time spent in each state:
Room     13326
Hall      6550
Lobby     3213
Exit       994
Name: State, dtype: int64
Room     13.326
Hall      6.550
Lobby     3.213
Exit      0.994
Name: State, dtype: float64
Average time through system: 24.083
       Min   Max    Mean  Median     StdDev
Room   1.0  76.0  13.326     9.0  12.610302
Hall   1.0  37.0   6.550     5.0   5.938813
Lobby  1.0  18.0   3.213     2.0   2.682095
Exit   0.0   1.0   0.994     1.0   0.077227
       Min     Max    Mean  Median      StdDev
```

Room 20.0 1520.0 266.52 180.0 252.206046
Hall 35.0 1295.0 229.25 175.0 207.858455
Lobby 30.0 540.0 96.39 60.0 80.462836
Exit 0.0 10.0 9.94 10.0 0.772269

The results will be different for different seeds, and the simulation can be random if the seed value is -1.