2.1 Create a Simulator

To verify my simulation code, I will break down my code based on every given requirement in the instructions.

- "You wish to model one logical-day (24 logical-hours) of simulation time."
    - To model this, I implemented a while loop that runs while current time is less than the total time allowed (24 hours). Since the input of riders is per minute (from 2.2), I set the total time to be 24 hours in minutes (24*60), however for more precision, I incremented the current time by 0.01.
- "There are some number of stations, m … modeled as elementary queue"
    - To model a station, I created a BikeStation object with two variables: a rider queue count and a bike count. For the queue, the incoming rider increments the counter, and it is decremented when a rider uses a bike.
- "The n riders arrive randomly… distributed exponentially with mean rate lambda"
    - To implement this, I incremented a newRider counter by the np.random.exponential output. Since the average is per minute, I made it so newRider is incremented only every whole minute of simulation time. Once newRider reaches a whole number, this rider is assigned to a random station.
- "When a rider arrives, she selects a bike station randomly…"
    - For this, I implemented a list of probabilities pulled from the start_station_probs.csv file. This list was then used to choose a random integer with np.random.choice, since each integer could be assigned a probability. Each integer relates to a station.
- "Rider goes to station i. If a bike is available, she takes it … otherwise waits. Allow for unlimited bikes…"
    - To allow for unlimited bikes, there is no limit to the bike count variable inside the BikeStation object. For using bikes or going to the queue, the simulator automatically adds a rider to the queue by default, and then checks if a bike is there (both of these are methods inside the BikeStation object). If a bike is there, the queue decreases and the bike counter decreases. If a bike isn't, nothing changes (since the rider is already added to the queue).
- "Rider chooses destination randomly"
    - To model this, I used a 2D array (81x133) where the 81 are the starting locations and 133 are the destination locations. This means that each column is the probabilities to go from the starting location (which is the index of the column). This column was used with np.random.choice to choose an integer relating to a station (these are methods). The count data is pulled from trip_stats.csv, and the column is converted to probabilities by dividing by the sum inside np.random.choice.

- “Rider uses a bike for some time… drawn from log-normal distribution”
    - To model this, I created a BikeRide object with a return time and a return location. The return location is pulled when the rider chooses a destination. The return time is modeled by np.random.lognormal added to the current time. The return of the bike is a method that checks if the return time is past the current time.

In addition to these instructions, I added a returnAll method that returns all bikes at the end of the day. Since the riders took these bikes before the simulation day ended, these successes/ wait times still count, and returning them all doesn’t affect this as the simulation ends. This method is just to ensure that there are no moving bikes at the end of the simulation, so that it is easier to see where they are returned.

2.2 A Baseline Experiment

The following inputs were provided to run the simulation with: 2 files that produce the starting probabilities and destination probabilities, riders arrive exponentially with mean 2.38, and ride times are log-normally distributed with mu=2.78 and sigma=0.619.

The simulation was run 10 times to produce the following 90% confidence interval (the wait time is given in minutes):

```
Success Probability Confidence Interval (90%):  (0.5553266035640266, 0.5713046989460877)
Wait Time Confidence Interval (90%):  (0.738679486237573, 0.7621753810733232)
```

The following is the output for every iteration of the simulation:

```
Total:  3393
Successful:  1966
Successful Rental Prob:  0.5794282346006484
Average Wait Time for Successful:  0.7324516785350966

Total:  3379
Successful:  1853
Successful Rental Prob:  0.5483870967741935
Average Wait Time for Successful:  0.7771181867242309

Total:  3262
Successful:  1902
Successful Rental Prob:  0.583077866339669
Average Wait Time for Successful:  0.7570977917981072

Total:  3492
Successful:  1891
Successful Rental Prob:  0.5415234822451317
Average Wait Time for Successful:  0.7615018508725542

Total:  3408
Successful:  1906
Successful Rental Prob:  0.5592723004694836
Average Wait Time for Successful:  0.7555089192025184
```

```
Total:  3410
Successful:  1897
Successful Rental Prob:  0.5563049853372434
Average Wait Time for Successful:  0.7590933052187665

Total:  3531
Successful:  2039
Successful Rental Prob:  0.5774568111016709
Average Wait Time for Successful:  0.7062285434036293

Total:  3386
Successful:  1878
Successful Rental Prob:  0.5546367395156527
Average Wait Time for Successful:  0.7667731629392971

Total:  3447
Successful:  1953
Successful Rental Prob:  0.566579634464752
Average Wait Time for Successful:  0.7373271889400922

Total:  3384
Successful:  1917
Successful Rental Prob:  0.5664893617021277
Average Wait Time for Successful:  0.7511737089201878
```