NYC Bike Share Simulation

1. Scenario

This project will involve the implementation of and experimentation with a discrete event simualtion of a bike sharing service in NYC. The simulation will involve riders and bike stations where riders can pick up and drop off bikes.

2.1 Simulator Implementation

This section will detail the implementation of the simulator. This simulator is implemented using three classes, Events, Stations, and Riders. Station and Rider objects will each hold data corresponding to bike stations and bike riders, respectively. Event objects will hold data corresponding to either an arrival of a rider at a station or the return of a rider to a station. Once at a station, a rider will either take or return a bike, or enter the correpsonding queue if necessary. A rider will need to enter a queue at a station if the station either does not have enough bikes for the rider that is arriving, or if it is at capacity when a rider returns. For this simulator, there will be m stations and n riders. the arrival times of riders will be given by an exponential distribution with mean rate λ. A rider will select station i to initially go to with probability p_i, and will return to station j with probability $q_{i,j}$ The duration of a rider's use of the bike will be generated using a log-normal distribution with mean μ and standard deviation σ.

In [ ]:
```
import numpy as np
import heapq
```

In [ ]:
```
class Event:
  def __init__(self, time, event_type, rider, source=None, destination=None):
    self.time = time
    self.event_type = event_type
    self.rider = rider
    self.source = source
    self.destination = destination

  def __lt__(self, other):
    return self.time < other.time

class Station:
  def __init__(self, id, initial_num, capacity):
    self.id = id
    self.num_bikes = initial_num
    self.capacity = capacity
    self.arrival_queue = []
    self.return_queue = []

class Rider:
  def __init__(self, id):
    self.id = id
    self.completed_ride = False
    self.arrival_time = None
    self.departure_time = None
    self.attempted_return_time = None
    self.return_time = None
    self.source = None
    self.destination = None
```

In [ ]:
```
def arrival(time, rider, arrival_rate, num_stations, arrival_probabilities, stations):
  time += np.random.exponential(1/arrival_rate)
  source = stations[np.random.choice(num_stations, p=arrival_probabilities)]
  rider.source = source
  rider.arrival_time = time
  return Event(time, "arrival", rider, source)

def attempt_return(time, rider, num_stations, duration_mean, duration_std, destination_probabilities, stations):
  duration = np.random.lognormal(mean=duration_mean, sigma=duration_std)
  time += duration
  destination = stations[np.random.choice(num_stations, p=destination_probabilities[rider.source.id])]
  rider.destination = destination
  rider.attempted_return_time = time
  return Event(time, "return", rider, rider.source, destination)
```

In [ ]:
```
def simulate(num_riders, duration_mean, duration_std, arrival_rate, num_stations,
             stations_initial, station_capacities, arrival_probabilities, destination_probabilities, sim_time, si
  stations = [Station(i, stations_initial[i], station_capacities[i]) for i in range(num_stations)]
  riders = [Rider(i) for i in range(num_riders)]
```

```
  event_list = []
  num_bikes_needed = stations_initial.copy()
  time = 0
  heapq.heappush(event_list, arrival(time, riders[0], arrival_rate, num_stations, arrival_probabilities, stations

  while time <= sim_time * 60 and len(event_list) > 0:
    current_event = heapq.heappop(event_list)
    time = current_event.time
    if current_event.event_type == "arrival":
      station = current_event.source
      rider = current_event.rider
      #print(f"Rider {rider.id} arrived at station {station.id}  at time {time}")
      if (rider.id < num_riders - 1):
        heapq.heappush(event_list, arrival(time, riders[rider.id + 1], arrival_rate, num_stations, arrival_probab
      if station.num_bikes > 0:
        #print(f"Rider {rider.id} departed station {station.id} at time {time}")
        rider.departure_time = time
        station.num_bikes -= 1
        heapq.heappush(event_list, attempt_return(time, rider, num_stations, duration_mean, duration_std, destina
      else:
        #print(f"Rider {rider.id} entered arrival queue at station {station.id} at time {time}")
        if (sim == 1):
          station.arrival_queue.append(rider)
        else:
          num_bikes_needed[station.id] += 1
          rider.departure_time = time
          heapq.heappush(event_list, attempt_return(time, rider, num_stations, duration_mean, duration_std, destin
      if len(station.return_queue) > 0:
        station.num_bikes += 1
        rider = station.return_queue.pop(0)
        rider.completed_ride = True
        rider.return_time = time
    else:
      station = current_event.destination
      rider = current_event.rider
      #print(f"Rider {rider.id} returned to station {station.id} at time {time}")
      if station.num_bikes < station.capacity:
        #print(f"Rider {rider.id} completed ride at {station.id} at time {time}")
        station.num_bikes += 1
        rider.completed_ride = True
        rider.return_time = time
      else:
        #print(f"Rider {rider.id} entered return queue at station {station.id} at time {time}")
        station.return_queue.append(rider)
      if len(station.arrival_queue) > 0:
        rider = station.arrival_queue.pop(0)
        #print(f"Rider {rider.id} departed station {station.id} at time {time}")
        rider.departure_time = time
        station.num_bikes -= 1
        heapq.heappush(event_list, attempt_return(time, rider, num_stations, duration_mean, duration_std, destina


  #Simulation data

  #End number of Bikes at each station
  #for station in stations:
    #print(f"Number of bikes at station {station.id}: {station.num_bikes}")

  #Average ride time for riders who obtained a bike and attempted to return it and number riders who got a bike
  return_time = 0
  num_successful_riders = 0
  for rider in riders:
    if (rider.departure_time is not None and rider.attempted_return_time is not None):
      return_time += (rider.attempted_return_time - rider.departure_time)
      num_successful_riders += 1
  return_time /= num_successful_riders
  #print(f"Average ride time: {return_time}")

  #Percent of successful riders
  average_riders = num_successful_riders / num_riders
  #print(f"Probability of successful ride: {average_riders}")

  #Average waiting time
  wait_time = 0
  for rider in riders:
    if (rider.departure_time is not None and rider.arrival_time is not None):
      wait_time += (rider.departure_time - rider.arrival_time)
  wait_time /= num_successful_riders
  #print(wait_time)

  return(average_riders, wait_time, num_bikes_needed)
```

Testing

This simulator was verified using several simulations with varying parameters and checking that the behavior matches the expected behavior

For example, in the following simulation the expected behavior is that each of the seven riders will arrive to one of the three stations, attempt to take a bike, and then depart the station. At some point in the future, the rider should return to a station and exit the system. Here, the intitial number of bikes at each station is enough so that no rider should have to enter the arrival queue to wait for a bike. Additionally, no rider should have to wait to return a bike since the capacities are high enough for these stations.

As can be seen in the output, this simulation results in expected behavior.

In [ ]:
```
num_riders = 7
num_stations = 3
duration_mean = 2.78
duration_std = 0.619
arrival_rate = 2.38
stations_initial = [7, 7, 7]
station_capacities = [10, 10, 10]
arrival_probabilities = [0.3, 0.2, 0.5]
destination_probabilities = [[0.1, 0.4, 0.5], [0.2, 0.5, 0.3], [0.3, 0.3, 0.4]]
sim_time = 5
results = simulate(num_riders, duration_mean, duration_std, arrival_rate, num_stations, stations_initial, station
```

```
Rider 0 arrived at station 0  at time 0.11050055988428756
Rider 0 departed station 0 at time 0.11050055988428756
Rider 1 arrived at station 0  at time 0.4011051696670064
Rider 1 departed station 0 at time 0.4011051696670064
Rider 2 arrived at station 2  at time 0.5985260212478244
Rider 2 departed station 2 at time 0.5985260212478244
Rider 3 arrived at station 2  at time 0.9797587889738364
Rider 3 departed station 2 at time 0.9797587889738364
Rider 4 arrived at station 0  at time 1.1471014307550351
Rider 4 departed station 0 at time 1.1471014307550351
Rider 5 arrived at station 2  at time 1.3981976820177298
Rider 5 departed station 2 at time 1.3981976820177298
Rider 6 arrived at station 1  at time 1.4891138879064436
Rider 6 departed station 1 at time 1.4891138879064436
Rider 5 returned to station 0 at time 6.5170584362084165
Rider 5 completed ride at 0 at time 6.5170584362084165
Rider 6 returned to station 0 at time 11.903136905732488
Rider 6 completed ride at 0 at time 11.903136905732488
Rider 2 returned to station 0 at time 11.983052607650501
Rider 2 completed ride at 0 at time 11.983052607650501
Rider 0 returned to station 2 at time 13.518148734676213
Rider 0 completed ride at 2 at time 13.518148734676213
Rider 3 returned to station 2 at time 19.39525371317459
Rider 3 completed ride at 2 at time 19.39525371317459
Rider 1 returned to station 0 at time 26.35014303234462
Rider 1 completed ride at 0 at time 26.35014303234462
Rider 4 returned to station 2 at time 35.850775974456894
Rider 4 completed ride at 2 at time 35.850775974456894
```

Here, the arrival and destination probabilites are varied so that every rider should arrive at station 0 (here, 0 corresponds to whichever the first station is), and then return to station 2. Since there are only five bikes at station 0, the last 2 riders will need to enter the arrival queue. Additionally, each rider will need to enter the return queue at station 2 since it is at capactity. Since no bikes arrive at station 0 and no bikes return to station 2, the riders in the queue will remain there.

In [ ]:
```
num_riders = 7
num_stations = 3
duration_mean = 2.78
duration_std = 0.619
arrival_rate = 2.38
stations_initial = [5, 5, 5]
station_capacities = [10, 10, 5]
arrival_probabilities = [1.0, 0.0, 0.0]
destination_probabilities = [[0.0, 0.0, 1.0], [0.2, 0.5, 0.3], [0.3, 0.3, 0.4]]
sim_time = 5
results = simulate(num_riders, duration_mean, duration_std, arrival_rate, num_stations, stations_initial, station
```

```
Rider 0 arrived at station 0  at time 0.17213467452149098
Rider 0 departed station 0 at time 0.17213467452149098
Rider 1 arrived at station 0  at time 1.0328058745771203
Rider 1 departed station 0 at time 1.0328058745771203
Rider 2 arrived at station 0  at time 1.1482384359637683
Rider 2 departed station 0 at time 1.1482384359637683
Rider 3 arrived at station 0  at time 1.4851933994158513
Rider 3 departed station 0 at time 1.4851933994158513
Rider 4 arrived at station 0  at time 2.742653378342511
Rider 4 departed station 0 at time 2.742653378342511
Rider 5 arrived at station 0  at time 2.8105074163012405
Rider 5 entered arrival queue at station 0 at time 2.8105074163012405
Rider 6 arrived at station 0  at time 2.9124955738769804
Rider 6 entered arrival queue at station 0 at time 2.9124955738769804
Rider 4 returned to station 2 at time 13.68322320737916
Rider 4 entered return queue at station 2 at time 13.68322320737916
Rider 2 returned to station 2 at time 14.361219051325019
Rider 2 entered return queue at station 2 at time 14.361219051325019
Rider 0 returned to station 2 at time 14.911963795611438
Rider 0 entered return queue at station 2 at time 14.911963795611438
Rider 3 returned to station 2 at time 23.199684361988314
Rider 3 entered return queue at station 2 at time 23.199684361988314
Rider 1 returned to station 2 at time 24.304938786573278
Rider 1 entered return queue at station 2 at time 24.304938786573278
```

In the process of creating this simulator, several basic simulations like the ones above were run to verify the simulator.

2.2 A Baseline Experiment

In this section, an experiment will be run using given data for starting station probabilities and the number of trips made between stations from June 2022. In this experiment, there will be 3500 riders. The arrival rate in the exponential distribution of rider interarrival times will be $\lambda = 2.38$ riders per minute, and the parameters for ride duration will be $\mu = 2.78$ and $\sigma = 0.619$. Note that these are the same parameters used in the testing simulations. Also, note that the print statements in the simulation code, which were used for testing, are commented out for this section.

This simulation will be used to estimate both percentage of successful riders, that is riders who obtained a bike, and the average waiting time amongst successful riders when each station starts with 10 bikes.

Locations will be imported from the given CSV files. Each location will be given an index corresponding to its position in start_station_probs.csv

In [ ]:
```
import csv
```

In [ ]:
```
locations = {}

with open('start_station_probs.csv', newline='') as csvfile:
    reader = csv.reader(csvfile)
    next(reader)
    for index, row in enumerate(reader):
        location = row[0]
        locations[location] = index

print(locations)
```

```
{'South Waterfront Walkway - Sinatra Dr & 1 St': 0, 'Grove St PATH': 1, 'Hoboken Terminal - Hudson St & Hudson P
l': 2, 'Hoboken Terminal - River St & Hudson Pl': 3, 'Newport Pkwy': 4, 'City Hall - Washington St & 1 St': 5, 'N
ewport PATH': 6, '12 St & Sinatra Dr N': 7, 'Hoboken Ave at Monmouth St': 8, 'Marin Light Rail': 9, 'Hamilton Par
k': 10, '14 St Ferry - 14 St & Shipyard Ln': 11, 'Liberty Light Rail': 12, 'Columbus Dr at Exchange Pl': 13, 'Har
borside': 14, '11 St & Washington St': 15, 'Washington St': 16, 'Sip Ave': 17, 'Hudson St & 4 St': 18, '8 St & Wa
shington St': 19, 'Madison St & 1 St': 20, 'City Hall': 21, 'Warren St': 22, 'Newark Ave': 23, 'Columbus Park - C
linton St & 9 St': 24, 'Grand St & 14 St': 25, 'Church Sq Park - 5 St & Park Ave': 26, 'Columbus Drive': 27, 'Van
Vorst Park': 28, 'Clinton St & Newark St': 29, 'Grand St': 30, 'Paulus Hook': 31, 'Manila & 1st': 32, '9 St HBLR
- Jackson St & 8 St': 33, 'Bloomfield St & 15 St': 34, '4 St & Grand St': 35, '7 St & Monroe St': 36, 'JC Medical
Center': 37, 'Clinton St & 7 St': 38, 'Willow Ave & 12 St': 39, 'Morris Canal': 40, 'McGinley Square': 41, 'Bruns
wick & 6th': 42, 'Jersey & 3rd': 43, 'Brunswick St': 44, 'Baldwin at Montgomery': 45, 'Adams St & 2 St': 46, 'Sou
thwest Park - Jackson St & Observer Hwy': 47, 'Marshall St & 2 St': 48, 'Journal Square': 49, 'Madison St & 10 S
t': 50, '6 St & Grand St': 51, 'Dixon Mills': 52, 'Lafayette Park': 53, 'Riverview Park': 54, 'Stevens - River Te
r & 6 St': 55, 'Mama Johnson Field - 4 St & Jackson St': 56, 'Pershing Field': 57, 'Hilltop': 58, 'Jersey & 6th S
t': 59, 'Essex Light Rail': 60, 'Monmouth and 6th': 61, 'Oakland Ave': 62, 'Adams St & 11 St': 63, 'Bergen Ave':
64, 'Fairmount Ave': 65, 'Montgomery St': 66, 'Christ Hospital': 67, 'Astor Place': 68, 'Heights Elevator': 69,
'Lincoln Park': 70, 'Leonard Gordon Park': 71, 'Communipaw & Berry Lane': 72, '5 Corners Library': 73, 'Glenwood
Ave': 74, 'Union St': 75, 'Dey St': 76, 'Jackson Square': 77, 'Bergen Ave & Stegman St': 78, 'Grant Ave & MLK D
r': 79, 'JCBS Depot': 80}
```

In [ ]:
```
arrival_probabilities = []

with open('start_station_probs.csv', newline='') as csvfile:
```

```
    reader = csv.reader(csvfile)
    next(reader)
    for row in reader:
        arrival_probabilities.append(float(row[1]))

print(arrival_probabilities)
```

```
[0.04467944609528249, 0.04350443784109227, 0.0336285420186836, 0.02983161451960613, 0.027034900658393056, 0.02635
514381712599, 0.025471459923478802, 0.024422692225523897, 0.023684670512148225, 0.023529297519858612, 0.022985492
046844958, 0.021616267552292723, 0.02025675386975859, 0.019285672667948495, 0.018664180698790032, 0.0176445454368
89432, 0.017265823768183496, 0.017032764279749073, 0.016955077783604264, 0.015750937093359747, 0.0154887451688710
21, 0.015216842432364194, 0.014605061275223834, 0.014304026102662704, 0.013381498960943114, 0.013216415156635398,
0.01294451242012857, 0.012866825923983763, 0.012565790751422634, 0.01233273126298821, 0.012128804210608092, 0.012
02198527840898, 0.012002563654372779, 0.011361650061178116, 0.01109945813668939, 0.01109945813668939, 0.011080036
512653189, 0.010905241896327371, 0.010759579716055857, 0.00999242556662588, 0.009963293130571578, 0.0099438715065
35376, 0.009526306589757035, 0.00938064440948552, 0.00937093359746742, 0.009341801161413118, 0.00933209034939501
6, 0.009283536289304511, 0.009060187612888189, 0.008846549748489968, 0.008700887568218454, 0.007972576666860883,
0.00753559012604634, 0.007487036065955835, 0.007438482005853305, 0.0073996387577929265, 0.007389927945774826, 0.
007360795509720523, 0.007253976577521412, 0.007224844141467109, 0.007108314397249898, 0.006632484608362951, 0.006
477111616073336, 0.0064674008040552355, 0.006302316999747519, 0.005622560158480452, 0.0056128493464623515, 0.0055
83716910408048, 0.0053409466099555245, 0.005321524985919323, 0.005311814173901221, 0.0051078871215211015, 0.00403
9697799529997, 0.0036124220707335545, 0.003301676086154324, 0.002728738177086368, 0.0026704733049777623, 0.001815
9218473848783, 0.001456621802715143, 0.0006894676532851677, 9.710812018100953e-06]
```

Using the indices for each location, a list is generated for return station probabilities, and each of these lists is stored in a list. Note that if an end location (such as any beginning with 6 ave in trip_stats.csv) is not in start_station_probs.csv, then it will not be considered.

In [ ]:
```
num_locations = len(locations)
trip_counts = np.zeros((num_locations, num_locations), dtype=int)

with open('trip_stats.csv', newline='') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        start_location = row['start']
        end_location = row['end']
        trip_count = int(row['count'])
        start_index = locations.get(start_location)
        end_index = locations.get(end_location)
        if start_index is not None and end_index is not None:
            trip_counts[start_index][end_index] = trip_count


total_trips_from_location = np.sum(trip_counts, axis=1, keepdims=True)
trip_probabilities = trip_counts / total_trips_from_location
destination_probabilities = list(trip_probabilities)

for i in range(num_locations):
    start_location = list(locations.keys())[i]
```

10 simulations will be run using the inputs described above, and the proportion of successful riders and their average waiting time will be output.

In [ ]:
```
num_riders = 3500
num_stations = len(arrival_probabilities)
duration_mean = 2.78
duration_std = 0.619
arrival_rate = 2.38
stations_initial = [10 for _ in range(num_stations)]
station_capacities = [10 for _ in range(num_stations)]
sim_time = 24
successful_riders = []
wait_times = []
num_sims = 15
simulate(num_riders, duration_mean, duration_std, arrival_rate, num_stations, stations_initial, station_capacities
for i in range(num_sims):
    results = simulate(num_riders, duration_mean, duration_std, arrival_rate, num_stations, stations_initial, sta
    successful_riders.append(results[0])
    wait_times.append(results[1])

print(successful_riders)
print(wait_times)
```

```
[0.9785714285714285, 0.9348571428571428, 0.9042857142857142, 0.9522857142857143, 0.9377142857142857, 0.9511428571
428572, 0.9588571428571429, 0.9597142857142857, 0.9497142857142857, 0.9374285714285714, 0.9794285714285714, 0.958
2857142857143, 0.978, 0.9597142857142857, 0.9645714285714285]
[7.4715643304506605, 5.4437697459491545, 16.3188318251254, 6.53590944752064, 4.842792106043185, 17.19768201286832
4, 12.795819474677076, 6.3611122173369745, 7.496433811884954, 12.017727753252924, 4.145581402791148, 8.1670657183
2926, 4.516017302003169, 5.241543382488506, 6.070263666961939]
```

Confidence Interval Estimation

Using the CLT and a student's t distribution, 90% confidence intervals will be estimated for each of the outputs in the above simulation.

In [ ]:
```
from scipy.stats import t
```

In [ ]:
```
def students_t_interval(data, confidence_level):
    sample_mean = np.mean(data)
    sample_std = np.std(data)
    deg = len(data) - 1
    alpha = 1 - confidence_level
    t_value = t.ppf(1 - alpha/2, deg)
    margin_of_error = t_value * (sample_std / np.sqrt(len(data) - 1))
    confidence_interval = (sample_mean - margin_of_error, sample_mean + margin_of_error)
    return confidence_interval
```

In [ ]:
```
print(f'90% Confidence Interval for the Proportion of Successful Riders: {students_t_interval(successful_riders, (
print(f'90% Confidence Interval for the Average Wait Time of a Successful Rider: {students_t_interval(wait_times,
```

```
90% Confidence Interval for the Proportion of Successful Riders: (0.9446648018805741, 0.9626113885956162)
90% Confidence Interval for the Average Wait Time of a Successful Rider: (6.37972468741404, 10.2365572056104)
```

2.3 An Idealized Experiment

In this section, the simulator will be used to determine the minimum number of bikes needed to meet demand fully. Here, meet demand fully will mean that the average wait time will be zero. Note that since the interarrival time of riders is 2.38 riders per minute, it is impossible to guarantee that all riders will receive a bike, nor is it typical that every rider will recieve a bike (since 2.38/minute is 3427 riders over the 24 hour period the simulation is run).

This experiment will be done by running the simulation 50 times, and keeping track of the number of bikes needed at each station to prevent any riders from needing to wait. Then, the max number of bikes for each station will be used, since this gaurantees that over all of the previous simulations no riders would have entered the arrival queue.

In [ ]:
```
num_riders = 3500
num_stations = len(arrival_probabilities)
duration_mean = 2.78
duration_std = 0.619
arrival_rate = 2.38
stations_initial = [0 for _ in range(num_stations)]
station_capacities = [np.inf for _ in range(num_stations)]
sim_time = 24
num_sims = 50
bikes_needed = np.zeros((num_sims, num_locations), dtype=int)
for i in range(num_sims):
    results = simulate(num_riders, duration_mean, duration_std, arrival_rate, num_stations, stations_initial, sta
    bikes_needed[i, :] = results[2]

max_values = np.max(bikes_needed, axis=0)
num_bikes_result = np.array(max_values)
print(num_bikes_result)
```

```
[45 37 39 37 35 29 30 23 33 34 40 32 19 34 32 24 20 28 23 29 32 24 18 23
 24 22 21 20 18 18 25 27 19 19 22 19 17 18 23 18 18 24 21 19 19 20 20 20
 13 22 15 19 16 16 18 13 14 28 20 16 18 19 19 18 19 16 13 16 15 14 11 13
 15 16 15  8  9  8  5  5  0]
```

As can be seen in the below code, the above values for the number of intial bikes at each station gives a wait time of 0 minutes for the 3500 riders. Note that the randomness of the simulation makes it so these initial values do not guarantee that the wait time will always be 0.

In [ ]:
```
stations_initial = num_bikes_result
result = simulate(num_riders, duration_mean, duration_std, arrival_rate, num_stations, stations_initial, station_
print(f"Proportion of Successful Riders: {result[0]}")
print(f"Average Wait Time for Successful Riders: {result[1]}")
```

```
Proportion of Successful Riders: 0.98
Average Wait Time for Successful Riders: 0.0
```