OR 7230 project
Bluebike Deployment

Name: Jinqiao Chen; Ming Luo

1 Problem statement:

Bluebikes program is Metro Boston's public share program that was founded in the fall of 2007. Until now, it has been expanded to be an important part of Boston's transportation network with over 4000+ bikes and more than 400+ stations across various towns such as Boston, Brookline, and Cambridge.

Similarly to other bike share systems, Bluebikes provides bikes that are locked into a network of docking stations in different areas around the city. People can unlock a bike from one station and return to another station (or the same one) that is in the Bluebikes network system.

The goal of our project is to analyze two of the most popular stations in two different regions and provide advice to optimize the usage of those stations and increase potential revenue.

These two stations are Forsyth St. At Huntington Ave station, which is located in Boston, and Mass Ave/Amherest station, which is located in Cambridge. There are some similarities between these two stations such as they are both in the center of popular campuses. Forsyth St. At Huntington Ave station sits in the center of Northeastern University and Mass Ave/Amherest station sits in the center of Massachusetts Institute of Technology. As given by the Bluebikes annual report, they are both the top 10 busiest stations in 2023. However, these two stations are different in other aspects as well. For instance, there are more residents around Forsyth St. At Huntington Ave station than Mass Ave/Amherest station since Forsyth St. At Huntington Ave station is located in a residential district while Mass Ave/Amherest station is in the business and tourist district.

There are specifically two cases we want to predict and prevent. First, is no bike available when people need to take bikes, or is no parking dock available when people need to end the trip? There are several consequences for this stockout:

- People will complain about the Bluebikes system which harms the brand. Fewer and fewer customers will continue to use Bluebikes and there might be no new customers as well. So this is a big loss to the company
- If we don't have the stockout, then people who need the bike will pay to ride bikes. But now, we couldn't get the money due to the unavailability of bikes. So it's a loss for us
- People who can't park their bikes will call customer service to complain and staff have to spend time and effort to solve the problems which could be avoided if we predict well and get some empty docks ready to park. So we can save such unnecessary labor costs if we can stop the stockout.

2 Data Collection

We collected the dataset from Blue Bikes' official website. They publish downloadable files of Bluebikes trip data each quarter and the comprehensive trip histories dataset we use is part of them. The information that is provided in the dataset includes but not limited to:

- each trip’s duration that counts in seconds
- start time and date, end time and date,
- Start Station Name & ID
- End Station Name & ID.

In addition, trips that are related to the staff services or inspections will be removed from the dataset. Or if people falsely start a trip or try to re-dock bikes (the length of the trip is less than 60 seconds), the related trip records will also be removed.

3 Birth and Death Process

The dataset we have gives us an intuition that the process of the bikes coming, and leaving the station could be a birth and death process with finite capacity N, which is a particular case of continuous-time Markov Chain problem. A continuous-time Markov Chain is a stochastic process that moves from one state to the other and the amount of time it stays in a state is exponentially distributed before moving to the next state.

For the Birth and death process, We can consider a bike station as a system where the state at any time is represented by the number of bikes in it at that time. For a specific station, if a bike arrives and is going to end its trip at this station, we see it as a birth. Similarly, if a bike leaves and is going to start a new trip at this station, we see it as a death. And for any given time t, the total number of arriving bikes by time t is the state of the process at time t, then the Poisson process is a continuous time Markov chain having states 0, 1, 2, 3, … N (N is the number of docks in one station) that always proceeds from state n to (n+1) or (n-1). And the state will always increase or decrease by 1 when a transition happens. In addition, we assume that new bike arrivals enter the station at a certain exponential rate and new bike departures at a certain exponential rate. These two rates don't have to be equal. What’s more, whenever we have certain bikes in the station, then the time until the next bike arrival is exponentially distributed with mean (1/arrival rate). This time duration is independent of the time until the next bike departure, which is also exponentially distributed with a mean (1/departure rate).

For the arrival rate and departure rate for each station, we can get them from checking the distribution of the interarrival time and inter-departure time. Each bike's arrival is independent from the other bikes’ arrival. This independence also applies to the bike's departure. And we hypothesize that every inter-time between two consecutive arrivals or departures is exponentially distributed. In addition, the number of docks is limited, which means that if we have an arrival, but no bike available, we will reject this customer.

4 Data Preprocessing

With the given dataset, we select the subset that contains all the trips between 4:30 and 6:30 pm from September 1st to 30th 2023. We pick peak commuting hours 4:30 - 6:30 pm specifically for several reasons.

- Since the two stations we picked are located at the center of two big schools, there will be a high demand for students who are going to go home after a whole day's class using bikes. In addition
- Forsyth St. At Huntington Ave station is close to the Ruggles station, an important station that provides buses, the Orange Line to Boston's nearby towns, and commuter rail to further areas. So there will be a high volume of people using the Bluebikes to either come to the Ruggles station or leave the station during the 4:30 - 6:30 pm, busy hours. In addition, Forsyth St. At Huntington Ave station is also close to the Green Line Northeastern station.
- Mass Ave/Amherest station is very close to the Charles River. From 4:30 - 6:30 pm, there will be a large volume of people who get off their work using blue bikes to do exercise along the bike line near the river. And the tourists will also use Blubikes to enjoy Charles River's sunset scene.

In addition, we pick September specifically because it is the month that students start to get back to school and the weather is perfect for biking. Thus, the data quality for September is good and large for us to analyze.

What's more, we also removed the weekend trip records whose distribution is significantly different from that of weekdays since the two stations we picked are both at the center of schools.

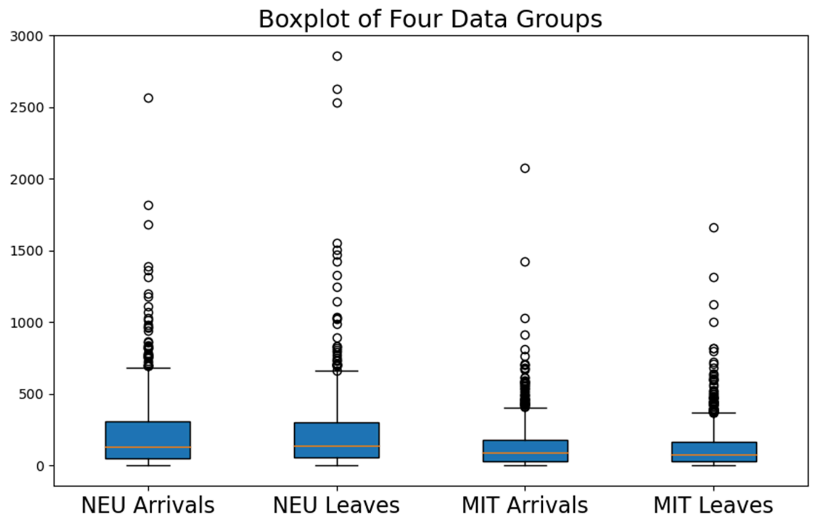
With the selected dataset, we calculated the interarrival time between each consecutive bike arrival and departure for both of the stations. Since the state (n) at any given time in the birth and death process can only go to its two neighboring states (n-1 or n+1), thus we assume that the bike leaves and comes once at a time because each state can only move to its neighbors in the birth and death process as we mentioned above. When we process the dataset, if several arrivals happen at the same time, we will just take one and remove the others. Similarly, if several departures happen at the same time, we will just take one and remove the others.

Thus, in the end, we developed two distinct datasets for each station:

- Death dataset: Time between consecutive bike departures

- Birth dataset: Time between consecutive bike arrivals

In addition, we checked the outliers for the four datasets: the death dataset for Forsyth St. At Huntington Ave station, the birth dataset for Forsyth St. At Huntington Ave station, the death dataset for Mass Ave/Amherest station, and the birth dataset for Mass Ave/Amherest station. Below is the side-by-side boxplot for the four datasets. From the boxplots we can see the distribution of the death dataset for Forsyth St. At Huntington Ave station is similar to the distribution of the birth dataset for Forsyth St. At Huntington Ave station, regardless of the outliers. Similarly, the distribution of the death dataset for Mass Ave/Amherest station is similar to the distribution of the birth dataset for Mass Ave/Amherest station, regardless of the outliers. However, the distribution of the death dataset for Forsyth St. At Huntington Ave station is different from that for Mass Ave/Amherest station. These observations also show that our data quality is good. For the outliers, we chose to remove them for better modeling.



5 Model & Validation

5.1 Goodness of fit

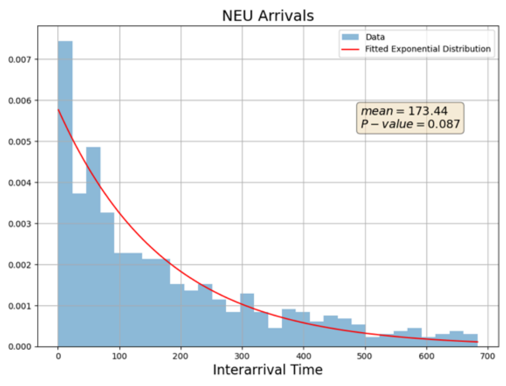
After we have cleaned the data and removed the outliers, we want to verify that if the four sets of data meet our hypothesis, i.e., whether they conform to an exponential distribution.

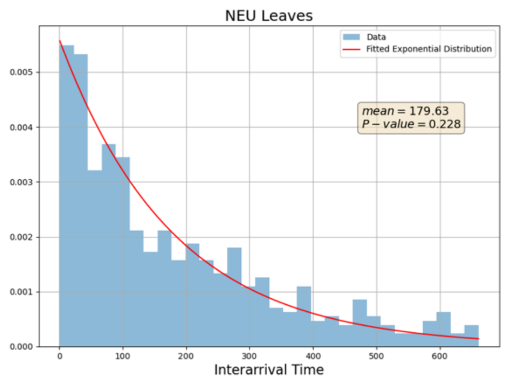
First, we use the maximum likelihood estimation (MLE) to calculate the parameters for each of the four data sets. And we get the following results:

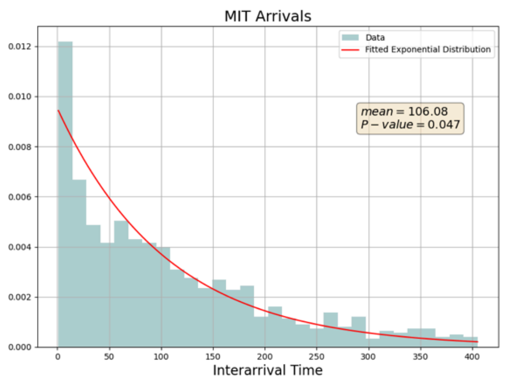
	NEU Arrival	NEU Leaves	MIT Arrivals	MIT Leaves
Parameters	1/173.44	1/179.63	1/106.08	1/93.95

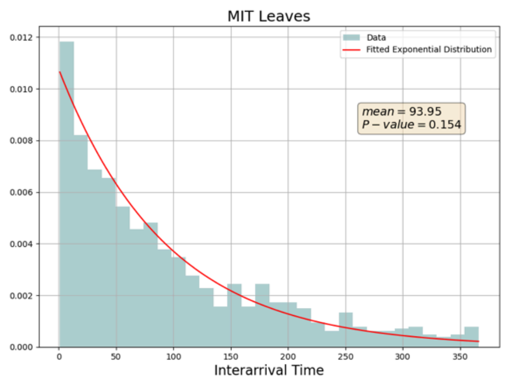
Next, we used the Kolmogorov–Smirnov test (K-S test) to verify if the data sets conform to the exponential distribution with the parameters we calculated. K-S test is a useful nonparametric test. It's widely used to determine if a sample comes from a specific probability distribution.

We applied this test to four data sets, then produced histograms and computed P-values for each of them. The results are shown as follow:









From the results, we can see that, except for the set of MIT Arrivals, the P-values of the other three data sets are greater than 0.05. Hence, we cannot reject the null hypothesis at a 95% confidence level, which means that all three data sets conform to the exponential distribution with the given parameters. As for the MIT Arrival data set, although its P-value is less than

0.05, it is only slightly different. Moreover, we can see from the histogram that the data fits the CDF of the exponential distribution very well, so we still think that this data set conforms to an exponential distribution with the given parameters.

5.2 Algorithm

Based on our findings, the results of the K-S test supported our hypothesis about the distribution of the data. Next, we proceeded to the algebraic part of the analysis.

We chose the Birth-Death process as the basis of our model. In this model, we denote state i as there are i bikes available at a dock station. Since each dock station has a fixed number of docks, the system has a capacity N. Then we can build up the balance equations, which are shown as follow:

$$\begin{aligned} i = 0, &\quad \lambda P_0 = \mu P_1 \\ 1 \le i \le N - 1, &\quad (\lambda + \mu) P_i = \lambda P_{i-1} + \mu P_{i+1} \\ i = N, &\quad \mu P_N = \lambda P_{N-1} \\ &\sum_{i=0}^{N} P_i = 1. \end{aligned}$$

After solving the equations, we can get:

$$P_i = \frac{\rho^i (1-\rho)}{1-\rho^{N+1}} \quad \left(i = 0, 1, 2, \ldots, N, \ \rho = \frac{\lambda}{\mu}\right)$$

P_i denotes the proportion of time that the system is in state i.

5.3 Criteria

Next, we will focus on how to evaluate the performance of the system. In the system, there are two states that deserve our attention:

1) State 0: No available bike at a dock station, which means users cannot borrow bikes when they have demands.
2) State N: All docks are occupied at a dock station, which means users are unable to return their bikes and end their trips.

Both states are "bad" states for this system. Because when the system is in these two states, it cannot meet customer's needs. Hence, our optimization goal is to reduce the proportion of time that the system is in these two states. If we express it algebraically, it should be as follow:

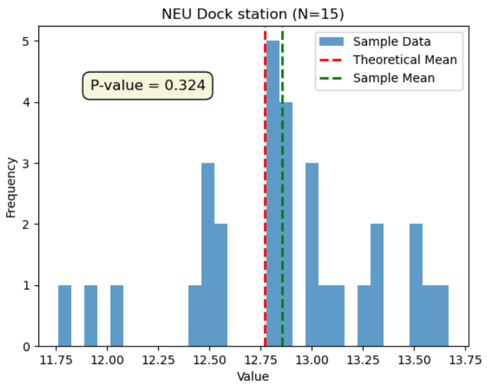
$$\left(P_0 + P_N\right) = \left[\frac{(1-\rho)\left(\rho^N + 1\right)}{1-\rho^{N+1}}\right]$$

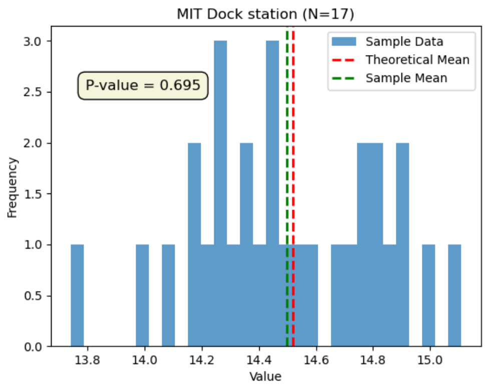
5.4 Verification

After getting the theoretical results, our next step is to verify whether the actual results are consistent with the theoretical model. A practical way for us to do this is to use simulation. On the one hand, it can save the time of collecting data; on the other hand, it can repeat the simulation many times and get more amount of data.

We used the SimPy package to perform the simulation process, which ran for a total of 100,000 minutes each time. The simulation will be performed 30 times to ensure the reliability of the analysis. The primary data collected from each simulation will be the proportion of time the system spends in state 0 and state N. The data collected will be used to determine the state of the system.

After collecting the data, we used the one-sample t-test to analyze the results. The purpose of the t-test is to compare the sample mean of the data collected from the simulation to the theoretical values. It will help us determine if there is a significant difference between the observed sample mean and the theoretical expected value. The results are shown as follow:





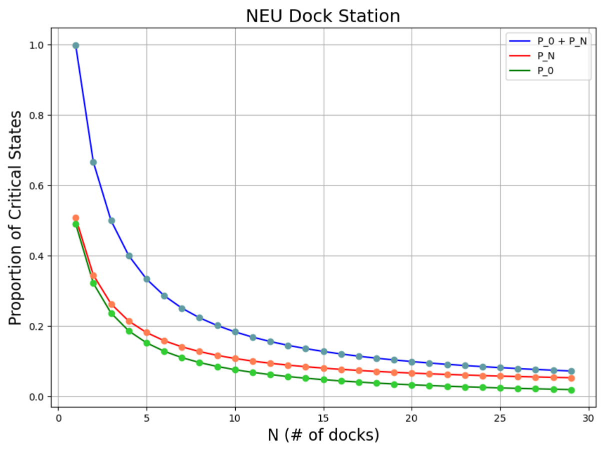
From the results, we can find that the P-value of both dock stations are greater than 0.05, which means there are not significantly difference between the simulation sample mean and the theoretical values.

6 Conclusion

6.1 Results of two docks

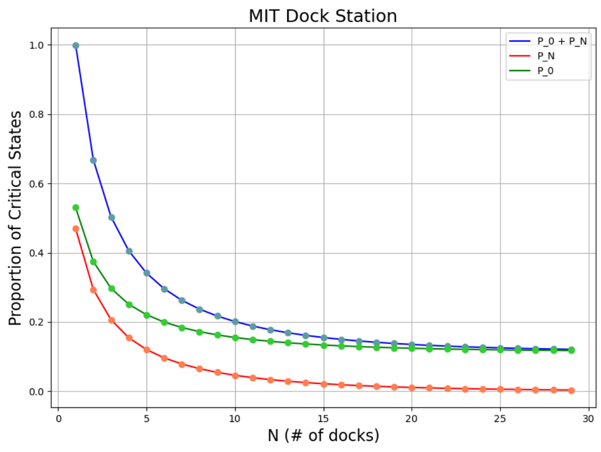
1) NEU Dock station:

N	$P_0 + P_N$	Diff
12	15.59%	
13	14.51%	1.08%
14	13.58%	0.93%
15	12.77%	0.81%
16	12.05%	0.72%
17	11.42%	0.63%
18	10.85%	0.57%



2) MIT Dock station:

N	$P_0 + P_N$	Diff
14	16.13%	
15	15.51%	0.62%
16	14.97%	0.54%
17	14.52%	0.45%
18	14.13%	0.39%
19	13.79%	0.34%
20	13.50%	0.29%



6.2 Analysis of the results

We graphed the function and observed the changes in proportion of $(P_0 + P_N)$ under different capacities. As we increased the capacity (N), we noticed that there is a consistent decrease in the proportion $(P_0 + P_N)$.

However, As N continued to grow, the rate of decrease in proportion slowed down, implying the existence of marginal effects. It means each additional unit of capacity contributed less to the reduction in $\left(P_0 + P_N\right)$ than the previous one.

This pattern is particularly insightful for decision-makers within the industry. Specifically, it shows the significance of appropriate strategy when we try to expand the capacity of a system. Decision-makers are faced with a question that how to balance the potential revenue of each additional dock against the financial cost of expanding capacity. This analysis offers a quantitative method for comparing potential revenue with the costs of adding capacity. It supports making wise decisions on investments.

6.3 Analysis of user pattern and station

Although the proportion of $(P_0 + P_N)$ at both stations decrease as N grows, there are still some differences. At NEU dock station, P_0 drops faster; while at MIT dock stations, P_N drops faster. It indicates that there may be two different user patterns:

1) NEU dock station:

   The customers have higher arrival rate. One potential reason is that more residential areas are around the station lead to higher bike arrivals during peak hours as residents return bikes and head home. What's more, NEU dock station is near by several subway stations, so bicycles become an important connection to the subway, resulting in NEU dock station reaching full capacity quickly.

   Hence, effective bicycle diversion strategies are necessary for peak hours crowd management.

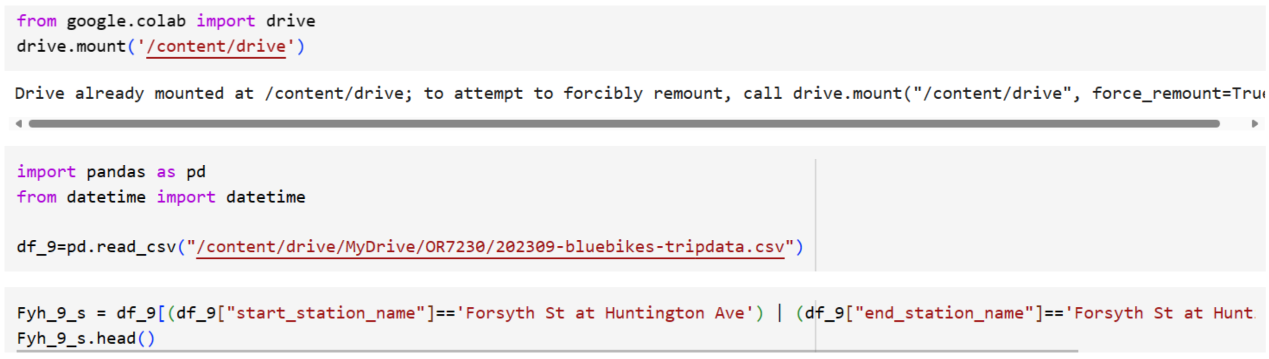
2) MIT dock station:

   The customers have higher departure rate. Because tourist areas along Charles River increases bike departures as users are not only students, but also tourists.

   Otherwise, due to the lack of subway options in the neighborhood, demand for bicycles increases during peak hours, requiring constant replenishment of bicycles to maintain service efficiency.

Appendix

- Dataset: [Bluebikes System Data | Blue Bikes Boston]()
- Code:
    - Data Preprocessing:



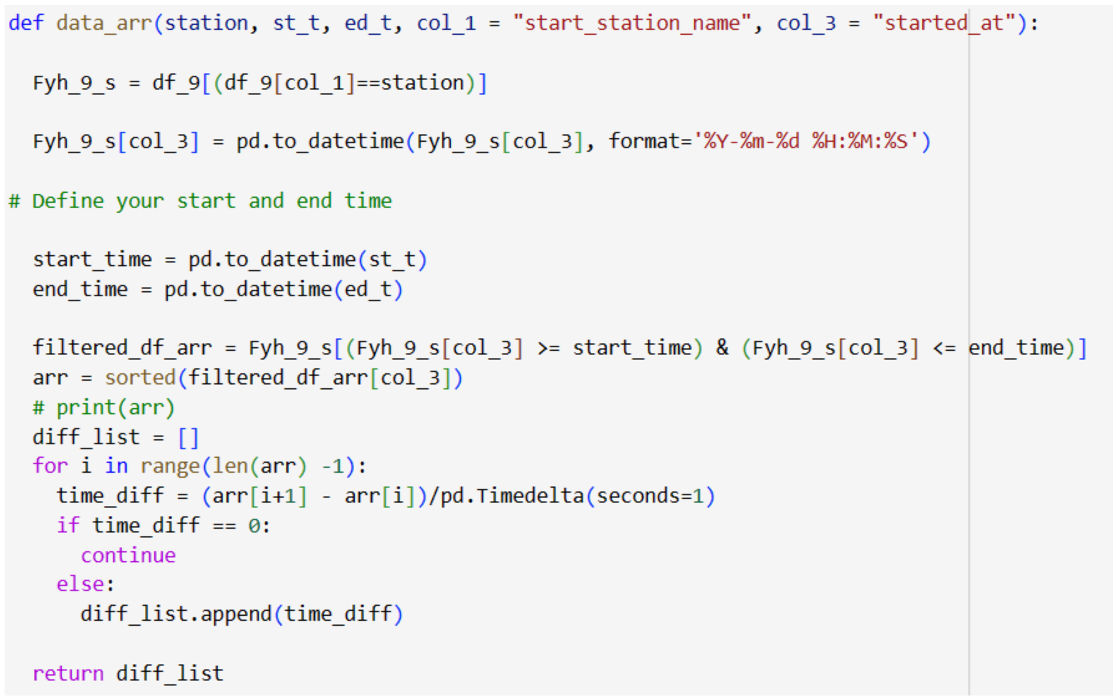
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tru
```

```
import pandas as pd
from datetime import datetime

df_9=pd.read_csv("/content/drive/MyDrive/OR7230/202309-bluebikes-tripdata.csv")
```

```
Fyh_9_s = df_9[(df_9["start_station_name"]=='Forsyth St at Huntington Ave') | (df_9["end_station_name"]=='Forsyth St at Hunt
Fyh_9_s.head()
```



```
def data_arr(station, st_t, ed_t, col_1 = "start_station_name", col_3 = "started_at"):

  Fyh_9_s = df_9[(df_9[col_1]==station)]

  Fyh_9_s[col_3] = pd.to_datetime(Fyh_9_s[col_3], format='%Y-%m-%d %H:%M:%S')

# Define your start and end time

  start_time = pd.to_datetime(st_t)
  end_time = pd.to_datetime(ed_t)

  filtered_df_arr = Fyh_9_s[(Fyh_9_s[col_3] >= start_time) & (Fyh_9_s[col_3] <= end_time)]
  arr = sorted(filtered_df_arr[col_3])
  # print(arr)
  diff_list = []
  for i in range(len(arr) -1):
    time_diff = (arr[i+1] - arr[i])/pd.Timedelta(seconds=1)
    if time_diff == 0:
      continue
    else:
      diff_list.append(time_diff)

  return diff_list
```

```python
day_list = []
# day_list_pass = [2,3,9,10,16,17,18,23,24,25,29,30]
day_list_pass = [2,3,9,10,16,17,23,24,30]
# df_name = []
for i in range(30):
  # df_name.append(f"bike_leave_intertime_day{i+1}")
  if (i+1) not in day_list_pass:
    # print(i+1)
    if (i+1)<10:
      day_list.append(str(0)+str(i+1))
    else:
      day_list.append(str(i+1))
```

```python
import csv
day_arr_all=[]
for day in day_list:
  s_d = f"2023-09-{day} 16:30:00"
  e_d = f"2023-09-{day} 18:30:00"
  day_arr_all = day_arr_all + (data_arr(station="Forsyth St at Huntington Ave", st_t = s_d, ed_t = e_d))
```

```python
import csv
day_arr_all_2 =[]
for day in day_list:
  s_d = f"2023-09-{day} 16:30:00"
  e_d = f"2023-09-{day} 18:30:00"
  day_arr_all_2 = day_arr_all_2 + (data_arr(station="MIT at Mass Ave / Amherst St", st_t = s_d, ed_t = e_d))
```

```python
def data_dep(station, st_t, ed_t, col_1 = "end_station_name", col_3 = "ended_at"):
  Fyh_9_s = df_9[(df_9[col_1]==station)]

  Fyh_9_s[col_3] = pd.to_datetime(Fyh_9_s[col_3], format='%Y-%m-%d %H:%M:%S')

# Define your start and end time

  start_time = pd.to_datetime(st_t)
  end_time = pd.to_datetime(ed_t)

  filtered_df_arr = Fyh_9_s[(Fyh_9_s[col_3] >= start_time) & (Fyh_9_s[col_3] <= end_time)]
  arr = sorted(filtered_df_arr[col_3])
  diff_list = []
  for i in range(len(arr) -1):
    time_diff = (arr[i+1] - arr[i])/pd.Timedelta(seconds=1)
    if time_diff == 0:
      continue
    else:
      diff_list.append(time_diff)

  return diff_list
```

```python
import csv
day_dep_all =[]
for day in day_list:
  s_d = f"2023-09-{day} 16:30:00"
  e_d = f"2023-09-{day} 18:30:00"
  day_dep_all = day_dep_all + (data_dep(station="Forsyth St at Huntington Ave", st_t = s_d, ed_t = e_d))
```

- K-S test and histogram:

```python
import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt




data = []

# Boxplot

plt.figure(figsize=(10, 6))

plt.boxplot(data, vert=True, patch_artist=True)

plt.title('Boxplot of Four Data Groups', fontsize=18)

plt.show()


Q1 = np.percentile(data, 25)

Q3 = np.percentile(data, 75)

IQR = Q3 - Q1


lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR

data_new = [i for i in data if upper_bound >= i >= lower_bound]


loc, scale = stats.expon.fit(data_new)


D, p_value = stats.kstest(data_new, 'expon', args=(loc, scale))

print("K-S test statistic:", D)

print("P-value:", p_value)

print("mean:", scale)


# Generate x values for the fitted exponential distribution curve
```

```python
x = np.linspace(min(data_new), max(data_new), 100)


# Plot the histogram and the fitted exponential distribution curve

plt.figure(figsize=(10, 7))

plt.hist(

   data_new,

   bins=30,

   density=True,

   alpha=0.5,

   label='Data',

   color='cadetblue')

plt.plot(x, stats.expon.pdf(x, loc=loc, scale=scale),

        'r-', label='Fitted Exponential Distribution')


# Adding labels and title

plt.xlabel('Interarrival Time', fontsize=16)

plt.title('MIT Arrivals', fontsize=18)


# Adding text box in the plot with mu and p-value

textstr = '\n'.join((

   r'$mean=%.2f$' % (scale,),

   r'$P-value=%.3f$' % (p_value,)))

props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)

plt.text(0.7, 0.75, textstr, transform=plt.gca().transAxes, fontsize=14,

        verticalalignment='top', bbox=props)


plt.legend()

plt.grid(True)

plt.show()
```

- Simulation:

```python
import simpy

import random




class BikeShareStation:

   def __init__(self, env, num_bikes, capacity):

       self.env = env

       self.dock = simpy.Container(env, init=num_bikes, capacity=capacity)

       self.time_no_bikes = 0

       self.time_full = 0

       self.last_update_time = 0


   def update_time_counters(self):

       now = self.env.now

       if self.dock.level == 0:

           self.time_no_bikes += now - self.last_update_time

       elif self.dock.level == self.dock.capacity:

           self.time_full += now - self.last_update_time

       self.last_update_time = now


   def return_bike(self):

       yield self.env.timeout(random.expovariate(60 / 106.08))

       self.update_time_counters()

       if self.dock.level < self.dock.capacity:

           yield self.dock.put(1)

       #     print(f'Bike returned at {env.now}. Total bikes:
{self.dock.level}')

       # else:

       #     print(f'Dock full at {env.now}. Cannot return bike.')
```

```
        self.update_time_counters()


    def borrow_bike(self):

        yield self.env.timeout(random.expovariate(60 / 93.95))

        self.update_time_counters()

        if self.dock.level > 0:

            yield self.dock.get(1)

            # print(f'Bike borrowed at {env.now}. Total bikes:
{self.dock.level}')

        # else:

        # print(f'No bike available to borrow at {env.now}.')

        self.update_time_counters()




def report_statistics(station):

    total_time = env.now

    print(

        f"Time with no bikes available: {station.time_no_bikes}, which is
{station.time_no_bikes / total_time:.2%} of "

        f"the total time.")

    print(f"Time with dock full: {station.time_full}, which is
{station.time_full / total_time:.2%} of the total time.")

    print(f"Critical Time Proportion: {(station.time_full +
station.time_no_bikes)/ total_time:.2%} of the total time.")




def borrower(env, bike_share_station):

    while True:

        yield env.process(bike_share_station.borrow_bike())




def returner(env, bike_share_station):
    while True:

        yield env.process(bike_share_station.return_bike())




env = simpy.Environment()

station = BikeShareStation(env, num_bikes=17, capacity=17)


env.process(borrower(env, station))

env.process(returner(env, station))


env.run(until=100000)

report_statistics(station)
```