Determine Bikes Allocations Between Stations using Policy Iteration

CS 504 Final Project Report

Chengyu Deng (cdeng@bu.edu)

This project is finished by Chengyu Deng. There are no other collaborators.

Please notice that this is the final report and the instruction of project 3 is in the README file **not** in this file.

Introduction

Bike riding is very popular in the Boston area in recent years. It is a fast, safe, and healthy way of commuting and exercising. The demand for the biking sharing service is huge in Boston area and biking sharing services in the Boston area like BLUE Bikes handle thousands of bike trips each day. However, although it is good that bike sharing services make riding a bike easy and affordable, such service still contains drawbacks which may hinder the usage of sharing bikes for users. In fact, the locations of BLUE bike's sharing stations are fixed, which means users have to rent/return bikes to those specific locations. In addition, the number of bike docks in the station is also fixed and hence users may face "no bike"/"docks full" situation when they want to rent/return a bike. In this project, I would like to simulate a bike moving solution between stations using an optimization algorithm called policy iteration to solve "docks full" and "no bike" problems and maximize bike utilization.

Data Sources

I used three datasets in this project, which are:

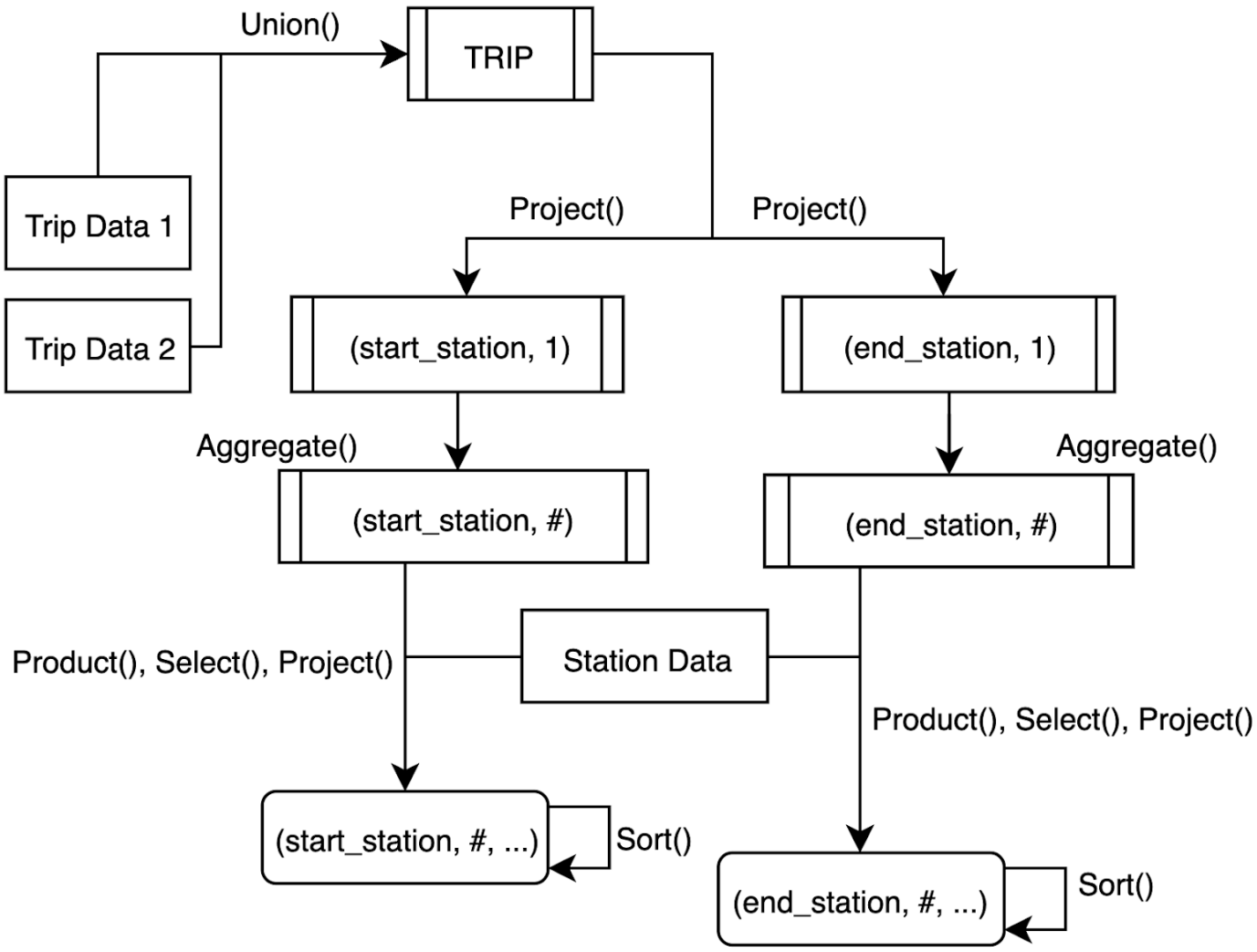
- `Hubway_Stations_as_of_July_2017.csv`
  This dataset indicates the total bike sharing locations mainly in Boston and Cambridge areas (some locations are in Somerville and Brookline). It contains important information such as the number of docks at that station, Latitude/Longitude data, street name, etc.
- `201801_hubway_tripdata.csv`
  This dataset maintains all the bike trip information in January 2018. It contains data such as trip length, start/end station, and its street name, user type, user gender, etc.
- `201802_hubway_tripdata.csv`
  This dataset contains all the bike trip information in February 2018. It contains data such as trip length,

start/end station, and its street name, user type, user gender, etc.

Please notice that `Hubway_Stations_as_of_July_2017.csv` is available through here. `201801_hubway_tripdata.csv` and `201802_hubway_tripdata.csv` are available through here. In my project, I uploaded all of the datasets to the http://datamechanics.io so that the program can automatically retrieve the datasets.

Data Processing

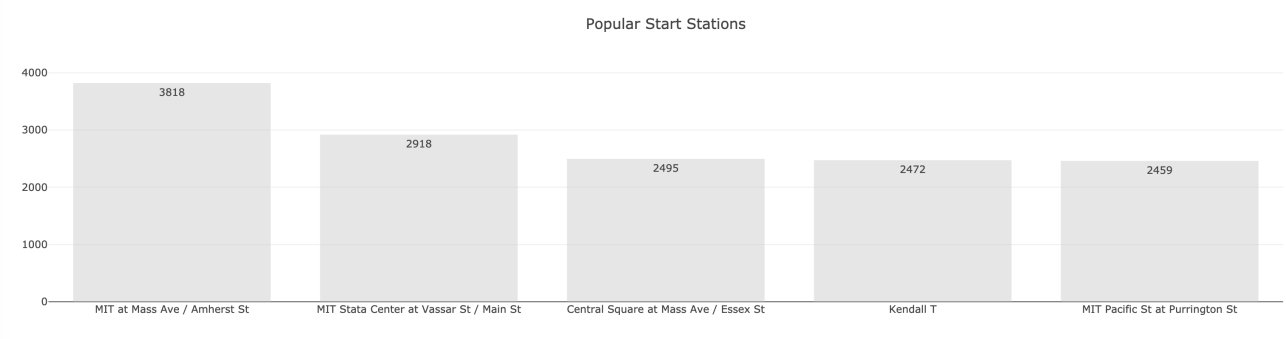
Before executing the optimization algorithm and the statistical analysis, the datasets need to be processed. First, we need to calculate the number of trips for each station based on the datasets and find out the popular stations. I use the relational building blocks available in MongoDB to process data. Then I sort the data to find out popular stations. Finally, the results of data processing are stored in MongoDB collections for further usages. The following diagram is the workflow of processing the datasets:



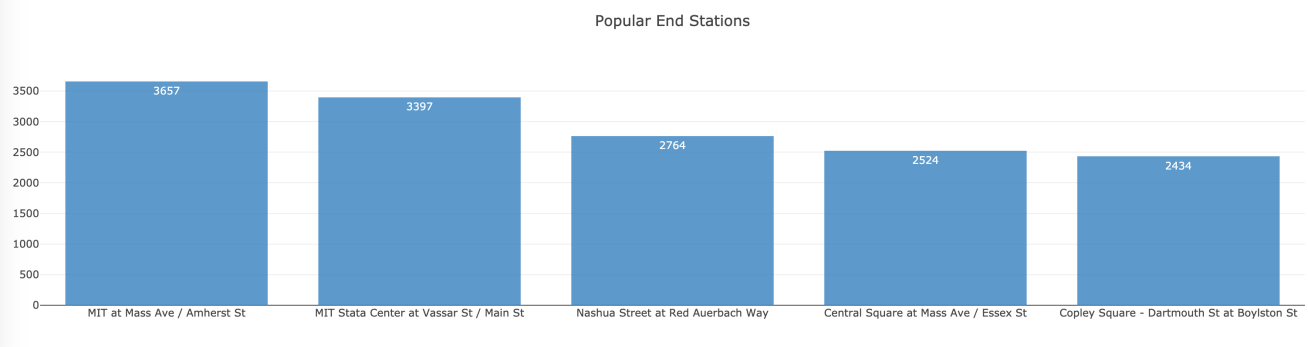
The original datasets will be loaded and stored in collections called `cdeng.stations_info` and `cdeng.bike_trip`. This happens in the `Project2_load_data.py` file. Then data process is finished in `Project2_most_popular_stations.py` file which will produce two collections called `cdeng.stations_popular_start` (stations with number of start trip from that station in descending order) and `cdeng.stations_popular_end` (stations with number of end trip to that station in descending order).

Here are some visualizations of the datasets and popular stations.

- Popular Start Stations chart (x: station name, y: number of starting trip from that station):

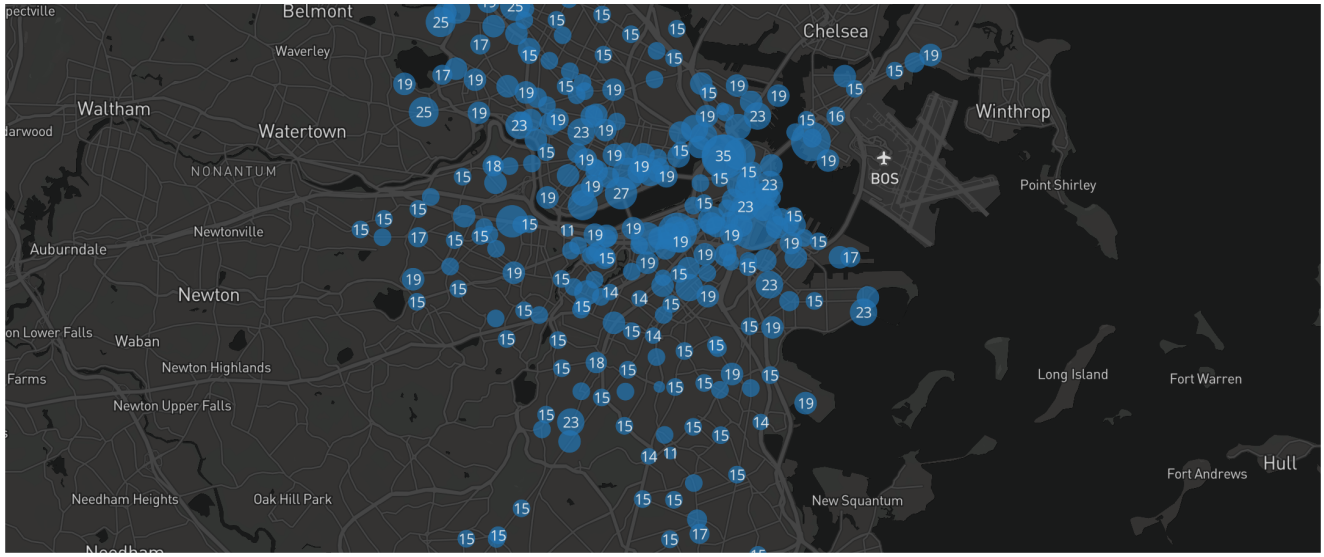


- Popular End Stations chart (x: station name, y: number of ending trip to that station):

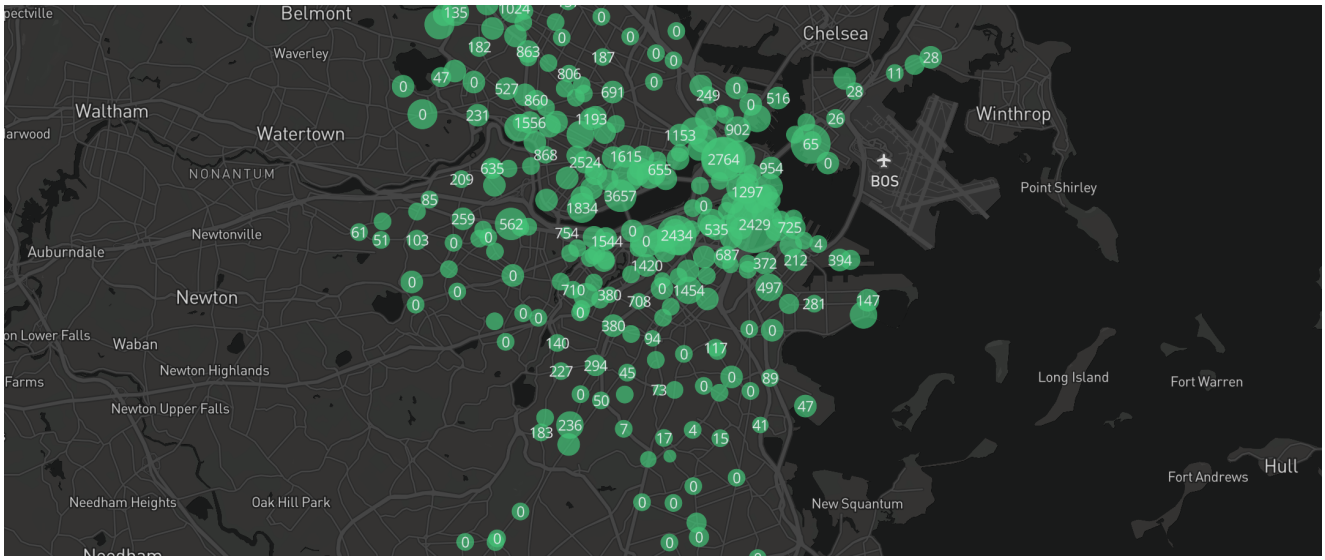


- A map based on docks (the circle markers indicate the locations of bike stations. The size of the circle marker depends on the number of docks. The number shown in the circle marker denotes the number of

docks):

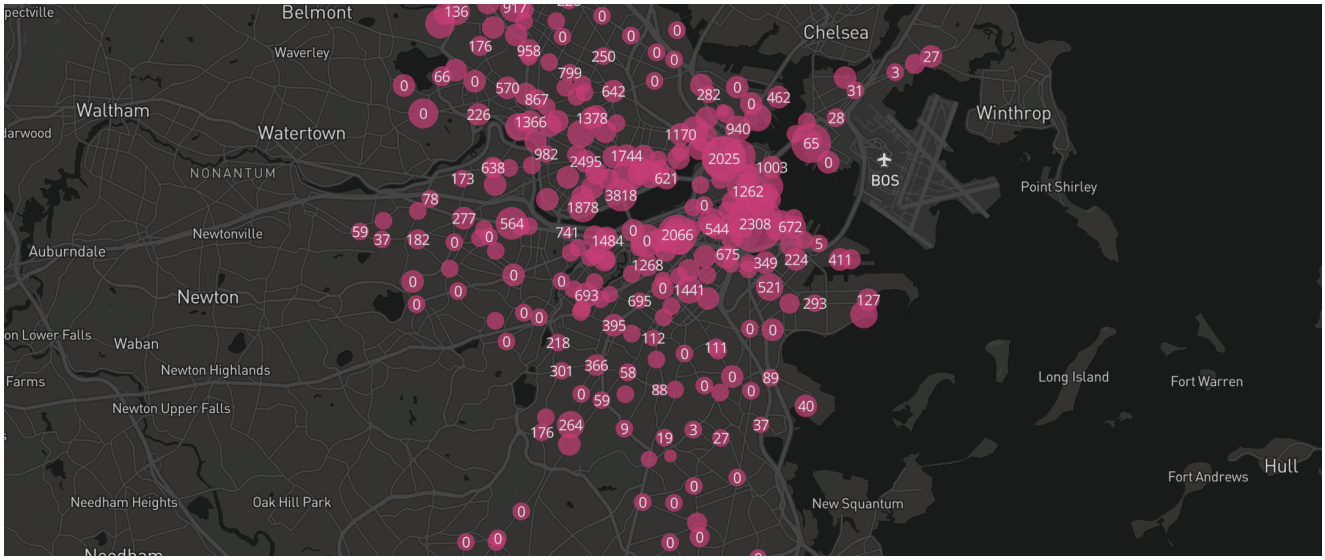


- A map based on the incoming rate (the circle markers indicate the locations of bike stations. The size of the circle marker depends on the number of docks. The number shown in the circle marker denotes total number of incoming bikes in January and Feburary, 2018 combined for that station)



- A map based on the outgoing rate (the circle markers indicate the locations of bike stations. The size of the circle marker depends on the number of docks. The number shown in the circle marker denotes total

number of outgoing bikes in January and Feburary, 2018 combined for that station)



Optimization Algorithm and Statistical Analysis

Policy Iteration Algorithm

Before introducing the algorithm we need to set up some assumptions:

- Select the most 2 popular stations for allocation optimization. (MIT at Mass Ave / Amherst St: station 1, MIT Stata Center at Vassar St / Main St: station 2)
- Move bikes from one station to the other every two hours during the day time (12 hrs, move bikes 6 times per day).
- A reward R is given for every bike rental.
- A cost C is given for moving a single bike.
- For each time, at most m bikes can be moved.
- The average bike incoming/outgoing rates for each station are calculated using the data from `cdeng.stations_popular_end` and `cdeng.stations_popular_start`. The number of incoming and outgoing bikes every two hours follows a Poisson Distribution.

The policy iteration algorithm using the a formalism from Markov Decison Process (MDP), which are the following:

- **State** s: A tuple that contains both stations' available bikes number.
- **Action**: Moving of n bikes ($0 < n \leq m$) at a state. Actions are in $[-n, n]$. In our project, $n = 5$. ($-n$ means moving n bikes from station 2 to station 1 and n means moving n bikes from station 1 to station 2)

- **State Transition Matrix**: Probability of a transition $\mathcal{P}_{ss'}$ from a state s to another state s'. It is based on the stations' incoming/outgoing bikes probability distribution.
- **Reward Function**: Total net rewards for move bikes at a state. (total rewards - total costs)
- **Gamma(γ)**: A discount factor from 0 to 1 for calculating future rewards in the main algorithm.

The policy ieration algorithm is an iterative algorithm and throughout the iteration the algorithm finds the optimizal state value which are define by the following:

- A policy π: an action for a certain state.
- State Value: An expected value for future reward at a state s at time t. We can write it as $V(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s]$.

To make it usable in the interation, we would rewrite it using Bellman Equation from dynamic programming and therefore the state value can have a recursive representation. We get:

- $V(s) = R_s + \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$ where s is the current state and s' is the next state. $\mathcal{S}$ is a collection of all states.

Once we have a recursion representation of state value, we can iteratively update the state value until convergence. This is the policy iteration algorithm and the pseudocode is the following:

Pseudocode of Policy Algorithm for bike allocations

```
Initialize policy(s)=0 and state value v(s)=0 for each state s;
do:
| do:
| | For each possible incoming rate/outgoing rate p, q:
| | | For each state s:
| | | | For each action:
| | | | | Update available bikes for each station to get s' and v(s');
| | | | | Based on the action, cost, and incoming rate, get net reward;
| | | | | Update current v(s) using the Bellman Equation of v(s);
| While (total change of state value > ε which is a small number);
| Greedy update of policy for each state;
While (policy is still changing);
```

I wrote the algorithm by myself because this algorithm is hard to implement using Z3 package. The details are in the `Project2_optimal_allocation.py` file. The limitation of this algorithm is that even though it uses

dynamic programming ideas already, it still has lots of loops going on for inspecting each possible situation. Therefore, the runtime of this algorithm is very slow.

Finally, there are two results of the policy iteration algorithm. The first result is the optimal policy matrix which represents actions based on state. The second result is the actual state value matrix after running the algorithm. The results are stored in the `cdeng.bike_allocation_strategy` collection in MongoDB.

Statistical Analysis

In this part, I calculated three correlation coefficients and p-values which are:

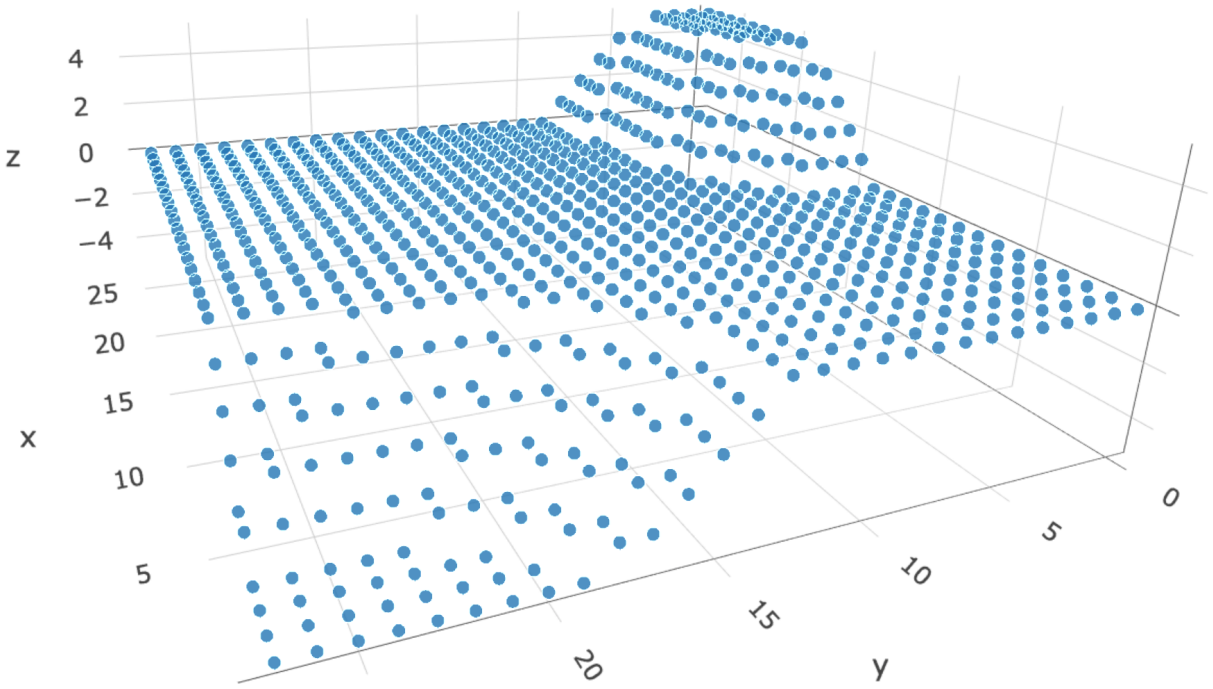
- Number of docks for each station and incoming bike trips.
- Number of docks for each station and outgoing bike trips.
- Incoming bike trips and outgoing bike trips for each station.

The statistical analysis result will be stored in the `cdeng.stations_dock_incoming_outgoing_stats` collection in MongoDB.

Results

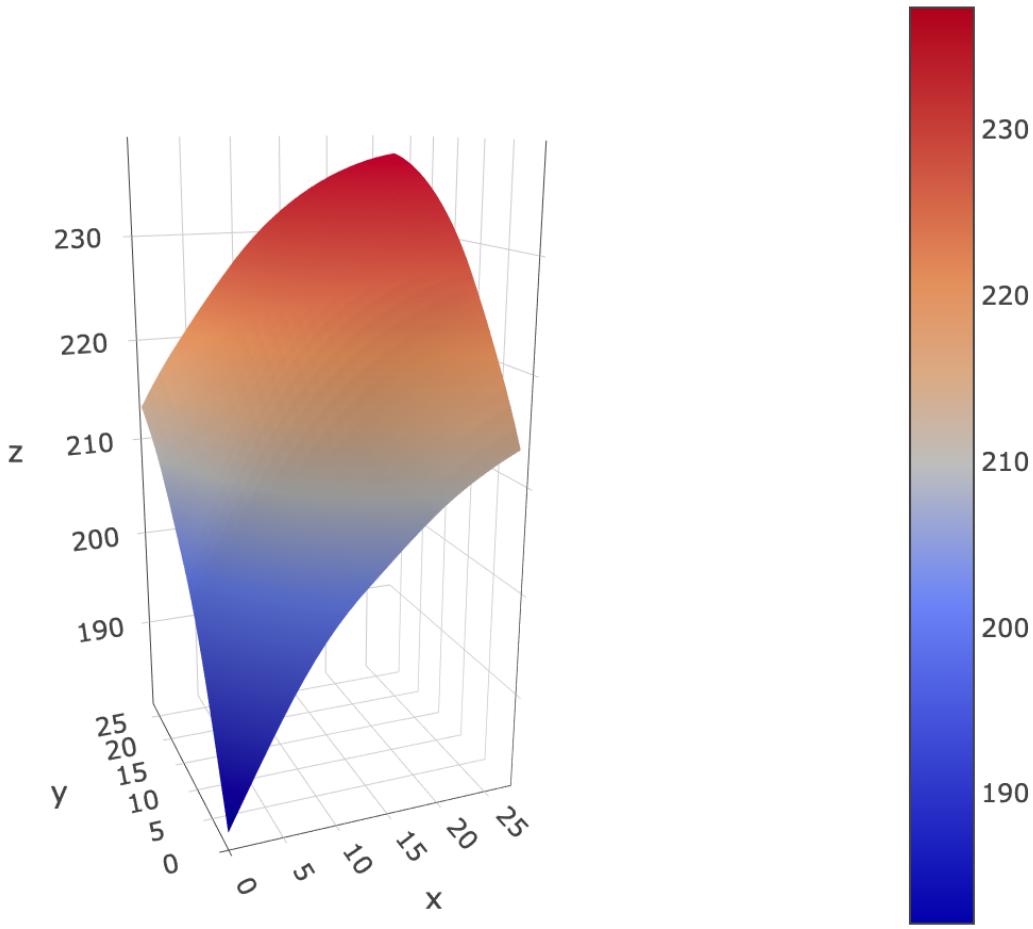
Here are the visualizations of results by the Policy Iteration Algorithm:

- Policy matrix based on number of bikes available in both stations:



The x and y axis denote the number of bikes at station 1 and station 2. The tuple (x, y) represents a state. The z axis indicates the policy option (z value is an integer and in $[-5,\ 5]$).

- State value matrix based on number of bikes available in both stations:



The x and y axis denote the number of bikes at station 1 and station 2. The tuple (x, y) represents a state. The z axis indicates state value.

- Statistical Analysis:

Cases	# of docks vs incoming trips	# of docks vs outgoing trips	Incoming trips vs outgoing trips
Corr Coefficient	0.4234	0.4508	0.9916
P-Value	0.0	0.0	0.0

P-values for those three cases are all 0, indicating that they are truly correlated based on the correlation coefficients that we calculated from the data.

Conlucsion and Future work

Policy Iteration returns an optimal solution for bike allocations for the two most popular stations. Also, after checking the p-values we are confident to claim that the number of docks does correlate to the incoming trip and outgoing trips. However, it is still an open question that how to find the optimal allocation strategy for all bike stations. It is challenging since the number of stations is quite large. One possible future work is to extend the algorithm to handle all stations. One potential solution is using a neural network to predict a state value instead of searching through all of them in the algorithm. In addition, the Poisson Distribution assumption needs to be examined using an advanced statistical method in the future.

Reference

https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/node20.html

https://en.wikipedia.org/wiki/Markov_decision_process

https://medium.com/@m.alzantot/deep-reinforcement-learning-demysitifed-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa

https://youtu.be/Nd1-UUMVfz4

https://www.bluebikes.com/system-data

http://cs-people.bu.edu/lapets/504/s.php#2

https://plot.ly/javascript/